



Master's Degree in Computer Engineering

# Security in Networked Computing Systems' notes

by Marco Micera

A.Y. 2016/2017

Available for free at [notes.altervista.org](http://notes.altervista.org)

# Messages structure overview

	Ciphers	Hash functions	MACs	SSL w/o MAC key	SSL	SSH	IPsec	DSS
<b>What is sent</b>	$E(e, m)$	$H(m)$	$S(a, m)$	$E(e, m \parallel H(m))$	$E(e, m \parallel S(a, m))$	$E(e, m)$	$E(e, m \parallel S(a, E(e, m))$	$S(\text{sk}, m) = \sigma$
<b>Kind of keys</b>	Symmetric	None	Symmetric	Symmetric	Both	Both	Both	Asymmetric
<b>Confidentiality or Secrecy</b> Protects data from eavesdroppers (spoofing).	Yes	No	No	Yes <i>No plaintext (m) sent over the network</i>	Yes	Yes	Yes	No <i>Everyone can decrypt the message with the sender's public key</i>
<b>Integrity</b> Guarantees that no tampering or alterations occur.	No	Yes	Yes	Yes <i>Hash sent with the message</i>	Yes	Yes	Yes	Yes
<b>Authentication</b> Ensures that only authorized senders and devices enter the network.	No	No	Yes <i>thanks to the pre-shared secret 'a' that kinda acts like a "key of an hash function"</i>	No	Yes	Yes	Yes	Yes
<b>Non-repudiability</b> The recipient can prove that the sender signed m and not m'.				No <i>The receiver can forge the message. Used in mutual-trust applications.</i>				Yes <i>Only the sender can produce its DS. Can be used with no mutual-trust.</i>
<b>Verifiability</b> The <u>recipient</u> can verify and prove that the sender, and no one else, signed the document.				Yes <i>If only the sender knows the pre-shared secret key 'a'</i>				Yes
<b>Non-forgability</b> The <u>sender</u> can prove that someone else has signed a message.								
<b>Public verifiability</b> Can anyone verify the signature?				No <i>Only the key holder can verify the message.</i>			Yes	
<b>Transferability</b> Can be transferred between users.				No <i>In a multiclient-server app where everyone share the same key, a user can create a new MAC</i>			Yes	

# Attacks and counter-attacks

sabato 03 giugno 2017 14:58

## Types of attacks

### 1. Ciphertext-only attack

The least strong.

- A ciphertext-only attack is one where the adversary (or cryptanalyst) tries to deduce the decryption key or plaintext by only observing ciphertext. Any encryption scheme vulnerable to this type of attack is considered to be completely insecure.

### 2. Known-plaintext attack

The adversary knows pairs of  $(p, c)$ .

S/he can then perform brute-force attacks against the key based on those pairs.

- A known-plaintext attack is one where the adversary has a quantity of plaintext and corresponding ciphertext. This type of attack is typically only marginally more difficult to mount.

### 3. Chosen-plaintext attack

The attacker can choose plaintexts and get corresponding ciphertext.

- "Get": from a server, or by his/her self.
- A chosen-plaintext attack is one where the adversary chooses plaintext and is then given corresponding ciphertext. Subsequently, the adversary uses any information deduced in order to recover plaintext corresponding to previously unseen ciphertext.

### 4. Adaptive chosen-plaintext attack

Not only the adversary can choose messages, but next ones depends on previous ones

- An adaptive chosen-plaintext attack is a chosen-plaintext attack wherein the choice of plaintext may depend on the ciphertext received from previous requests.

## Forgery types

### 1. Selective forgery

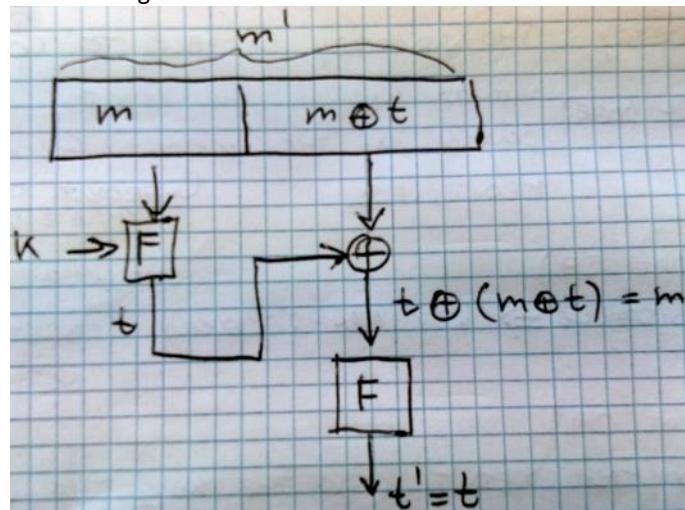
The adversary chooses the messages and then forges (creates) the tag.

### 2. Existential forgery (weaker)

The adversary has no control over the message so the message  $m$  could be **meaningless**.

Two types:

- Find  $m' \neq m: h(m') = h(m)$ , without knowing the key.
  - Example: in the CBC-MAC construction w/o the last encryption,  $m$  and  $m \parallel (m \oplus t)$  produce the same tag.

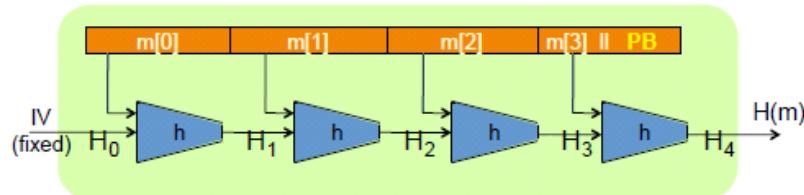


- Computing  $(\text{message}, \text{tag})$  pairs, without knowing the key.

- Example: in HMAC, starting from a  $[m, S(k, m)]$  pair, the adversary can compute the  $[m \parallel \text{padding} \parallel w, h(w, t)]$  pair, where  $w$  is a block.

Proof is obvious just by looking at the Merkle-Damgård hash construction (it's just a

continuation of this scheme):



## Confusion and diffusion

Shannon said that a cipher, in order to be secure, has to implement **both** of the following properties

- **Confusion:** the relationship b/w key and CT is obscured
  - Subsets of bits get **substituted**
  - Both AES and DES use confusion
- **Diffusion:** the influence of one PT symbol is spread over many CT symbols with the goal of hiding statistical properties of the PT
  - **Permutation**
  - Both AE and DES use diffusion
  - E.g.: by changing one plaintext bit, half of the ciphertext bits are usually changed.  
The cipher's output has to look like a random variable.

When those two properties are concatenated, a **product cipher** is obtained.

# Symmetric encryption

mercoledì 01 marzo 2017 19:38

- ★ • Confidentiality

## Cipher

- **Definition**

A cipher defined over  $(K, P, C)$  is a pair of "efficient" algorithms  $(E, D)$  where:

- $E: P \times K \rightarrow C$
- $D: C \times K \rightarrow P$

such that the operations are invertible:

- $\forall p \in P, k \in K: D(E(k, p)) = p$

- **Security**

Given  $C$  and  $P$ , it is difficult to determine  $K$ , unless it is used just once.

## Perfect secrecy

- **Definitions**

- **Kerchoff's principle:** "A cryptosystem should be secure even if everything about the system, except the key, is public knowledge"
- **Perfect secrecy by Shannon**

A cipher  $(E, D)$  defined over  $(K, P, C)$  has perfect secrecy if

- $\forall p \in P, c \in C: Pr(P = p | C = c) = Pr(P = p)$

The a posteriori probability (given the fact that the adversary has read the ciphertext), is equal to the a priori probability (the second one).

- **Another definition**

A cipher  $(E, D)$  defined over  $(K, P, C)$  has perfect secrecy if

- $\forall m_1, m_2 \in P(|m_1| = |m_2|)$ , (for each pair of messages having the same length)

$\forall c \in C$ ,

$$Pr(E(k, m_1) = c) = Pr(E(k, m_2) = c)$$

where  $k \xleftarrow{\text{random}} K$

- **Theorems**

- **Shannon's theorem:** In a perfect cipher,  $|K| \geq |P|$

▪ Proof by contradiction

- $\begin{cases} |K| < |P|: \text{contradiction} \\ |C| \geq |P|: \text{otherwise encryption is not invertible} \end{cases} \Rightarrow |C| > |K|$

▫ Selecting one message  $p^*$  with  $Pr\{P = p^*\} \neq 0$ , we start encrypting  $P^*$  with all the possible keys.

- $|C| > |K| \Rightarrow \exists c^* \in C \text{ s.t. } Pr\{P = p^* | C = c^*\} = 0 \neq Pr\{P = p^*\}$

◆ There exists at least one ciphertext which is not an image of  $p^*$

- $\exists p^*, c^* \text{ s.t. } \text{Shannon's equality does not hold} \Rightarrow \text{cipher is not perfect} \Rightarrow \text{contradiction}$

- Unconditional security necessary conditions

- i. Key bits are truly **randomly** chosen
- ii.  **$|K| \geq |P|$**  (Shannon's theorem)

- **One Time Pad (OTP)**

- **Definition**

- $m \in \{0,1\}^t$
  - $k \in \{0,1\}^t$ 
    - **One-time**, pre-shared
    - Truly random chosen
    - As long as the message
  - $E(k, m) = m \oplus k$
  - $D(k, c) = c \oplus k$

- Pros and cons

Pros	Cons
<ul style="list-style-type: none"><li>• Perfect (unconditional security)</li><li>• Very fast enc/dec</li><li>• Only one key maps <math>m</math> into <math>c</math></li></ul>	<ul style="list-style-type: none"><li>• Key as large as the message (if I'm able to pre-share a <math>t</math>-bit key, I'm able to pre-share the message itself)</li><li>• <b>One-time key:</b> two-time pad has the following problem<ul style="list-style-type: none"><li>• <math>c_1 = m_1 \oplus k_1, c_2 = m_2 \oplus k_2</math></li><li>• <math>c_1 \oplus c_2 = m_1 \oplus m_2</math> releases information due to ASCII redundancy</li></ul></li><li>• Know-plaintext attack breaks OTP: <math>k = m \oplus c</math><ul style="list-style-type: none"><li>• Malleable: perturbation on ciphertexts have predictable impact on decrypted plaintexts<ul style="list-style-type: none"><li>◦ <math>p' = c' \oplus k = c \oplus r \oplus k = p \oplus k \oplus r \oplus k = p \oplus r</math></li><li>◦ The attacker should then also know the relative plaintext</li></ul></li></ul></li></ul>

- OTP perfect secrecy

- OTP has perfect secrecy iff:

1) $\forall m, m' \in P: \text{len}(m) = \text{len}(m')$	All messages have same length
2) $\forall m \in P: \Pr(M = m) \neq 0$	There are no impossible messages
3) $k \xleftarrow{\text{random}} K$	Key is random

- Proof

- Proof target:  $\Pr(P = p | C = c) = \Pr(P = p)$  (perfect secrecy theorem)
- Bayes' theorem to the first operand:  $\Pr(P = p | C = c) = \frac{\Pr(P=p, C=c)}{\Pr(C=c)}$

- ◆ Numerator evaluation

- ◆ Lemma: there's only one key that maps  $p$  into  $c$ , that is  $k = p \oplus c$
- ◆  $\Pr(P = p, C = c) = \Pr(P = p, C = c, K = k) = \Pr(P = p, K = k)$
- ◆ Those events are independent:

$$\Pr(P = p, K = k) = \Pr(P = p) \cdot \Pr(K = k) = \Pr(P = p) \cdot \frac{1}{2^t}$$

- ◆ Denominator evaluation

$$\Pr(C = c) = \frac{\text{number of } (p, k) \text{ pairs that produce } c}{\text{number of all possible } (p, k) \text{ pairs}} = \frac{|P|}{|P| \cdot |K|} = \frac{1}{|K|} = \frac{1}{2^t}$$

- ◆ Notice that for each  $(p, k)$  pair there's only one ciphertext produced.

- ◆ By dividing them, the theorem has been proved.

- XOR property

- $p$  random variable on  $\{0,1\}^n$
- $k$  independent uniform (truly random chosen) variable on  $\{0,1\}^n$
- $c = p \oplus k$  is uniform on  $\{0,1\}^n$
- Proof with  $n = 1$

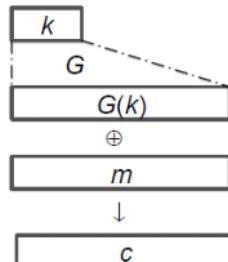
$p$	$k$	$c = p \oplus k$	$\Pr\{c\} = \Pr\{p\} \cdot \Pr\{k\}$
0	0	0	$\frac{1}{2} \cdot \Pr\{p = 0\}$
0	1	1	$\frac{1}{2} \cdot \Pr\{p = 0\}$
1	0	1	$\frac{1}{2} \cdot \Pr\{p = 1\}$
1	1	0	$\frac{1}{2} \cdot \Pr\{p = 1\}$

- $\Pr\{c = 0\} = \frac{1}{2} \cdot \Pr\{p = 0\} + \frac{1}{2} \cdot \Pr\{p = 1\} = \frac{1}{2} \cdot (\Pr\{p = 0\} + \Pr\{p = 1\}) = \frac{1}{2} \cdot 1 = \frac{1}{2}$
- Same thing for  $\Pr\{c = 1\} = \frac{1}{2}$

- Practical OTP

- Pseudo-Random Generator (PRG)

- Functioning: a shorter key  $k$  is a seed for a PRG that produces  $G(k)$ :

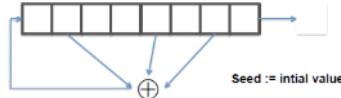


- Shannon's theorem says that this is not perfect.
- Encryption as secure as the PRG randomness.
  - ◆ There should be no efficient algorithm to distinguish a PRNG output from a TRNG one.
  - ◆ So called Cryptographically-Secure PRGs (CSPRGs)

- Badly made real One Time Pads

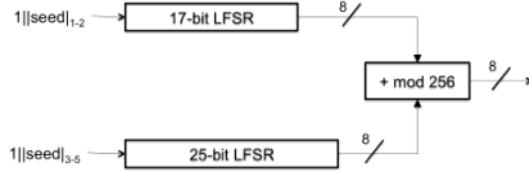
- Microsoft's PPTP used two-time pad
- 802.11b WEP (Wi-Fi)
  - Used to protect Wi-Fi network
  - The PRG had  $(IV \parallel key)$  as input.
    - ◆ The IV was only on 24 bits so after 12K frames there was a collision probability higher than 50%.
    - ◆ Even worse if the IV was a counter: on a netcard reboot, the IV resets to 0.
    - ◆ All these inputs use the same key: RC4 (the used PRNG) was weak w/ related keys.
      - ◆ Patch: different key for each frame.
      - ◆ RC4 is now deprecated.

- CSS (DVD encryption)
  - Its problem consisted in the PRNG
  - Used LFSRs (Linear Feedback Shift Registers)



- ◆ They are periodical after  $2^{\#bits+1}$  shift operations
- ◆ Attacks are  $O(\#bits)$ , predictable behaviour

- Key generation in CSS



- ◆ Key length is 5B, so this process is repeated 5 times
- ◆ Seed is also 5B, used to configure LFSRs like in figure

- How to break it

- ◆ First 20B of an mpeg known  $\Rightarrow \text{CSS\_keystream}|_{1-20} = \text{ciphertext}|_{1-20} \oplus \text{known\_plaintext}|_{1-20}$
- ◆ For every LFSR-17 initial state ( $2^{17}$ )
  - Run LFSR-17 and get its 20B output without the other 25-bit LFSR
  - $\text{CSS\_keystream}|_{1-20} - \text{LFSR17\_output} = \text{LFSR25\_output}|_{1-20}$  (candidate)
  - Check if two registers together produce  $\text{CSS\_keystream}|_{1-20}$

- Improvement: non-linear function instead of XOR

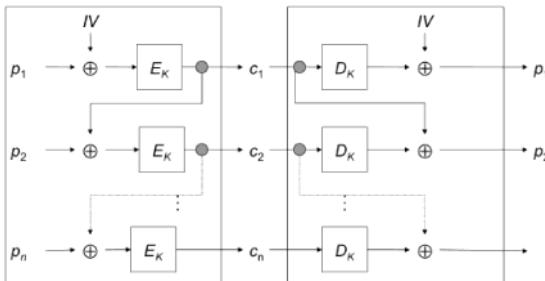
## Encryption modes

- Electronic codebook (ECB)
  - Blocks encrypted separately
  - Pros and cons

Pros	Cons
<ul style="list-style-type: none"> <li>• No block sync</li> <li>• No error propagation</li> <li>• Parallelizable</li> </ul>	<ul style="list-style-type: none"> <li>• Doesn't hide data pattern <math>\Rightarrow</math> traffic analysis</li> <li>• Allows ciphertext block re-ordering and substitution</li> </ul>

- Replay attack

- Cipher Block Chaining (CBC)
  - Scheme



- In formulas

Encryption	Decryption
$\begin{cases} CT_0 = IV \\ CT_i = E_e(PI_i \oplus CT_{i-1}), & i \in [1, t] \end{cases}$	$\begin{cases} CT_0 = IV \\ PT_i = CT_{i-1} \oplus D_e(CT_i), & i \in [1, t] \end{cases}$

- Initialization vector (IV) is a nonce: it has to be changed every time

- Pros and cons

Pros	Cons
<ul style="list-style-type: none"> <li>• <math>c_i</math> depends on all previous <math>p_i</math></li> <li>• ct-block reordering affects decryption</li> <li>• IV can be sent in clear: must not be tampered</li> </ul>	<ul style="list-style-type: none"> <li>• Error propagation</li> <li>• Bit error in <math>c_i</math> affects decryption of <math>c_i</math> and <math>c_{i-1}</math></li> <li>• <math>PT'_i = CT_{i-1} \oplus D_e(CT'_i)</math></li> <li>• <math>PT'_{i+1} = CT_i \oplus D_e(CT_{i+1})</math></li> </ul>

- PKCS#5 padding

# Block ciphers

venerdì 02 giugno 2017 15:26

## Fundamentals

- **$n = \text{block size}$**
  - A block cipher basically permutes all possible plaintexts to all possible ciphertexts
    - Number of permutations:  $(\#\text{plaintexts})! = 2^n!$ 
      - So a perfect key length should be  $k = \log_2 2^n! \sim (n - 1.44) \cdot 2^n \Rightarrow k \propto 2^n$ , [but  \$n \geq 64\$](#)
      - A key specifies a permutation
  - Shannon's: perfect if it is able to produce all possible permutations
  - Why cipher blocks input size  $n \geq 64 \text{ bits}$ 
    - Known-plaintext attack:  $(pt, ct)$  pairs
    - Attackers can build dictionaries
    - Dictionary size:  $\frac{2^n \cdot 2^n \text{ bits}}{8 \text{ Byte}} = \frac{1}{4} \cdot n \cdot 2^n \text{ Bytes}$
- |        |        |
|--------|--------|
| $pt_1$ | $ct_1$ |
| $pt_2$ | $ct_2$ |
| ...    | ...    |
| $pt_n$ | $ct_n$ |
- Width:  $2^n \text{ bits}$
  - Height:  $2^n \text{ bits}$
- Exhaustive key search theorem
    - Given  $\binom{k+1}{n}$  pairs of  $(pt, ct)$ , a key can be recovered by brute-force in  $O(2^{k-1})$
    - In DES and AES, two  $(pt, ct)$  pairs are enough
      - DES challenge: known the entire  $ct$  and the first three  $pt$  blocks, determine the key

## Lengths

DES	$n = 64 \text{ bits}$	$k = 56 \text{ bits}$
3DES	$n = 64 \text{ bits}$	$k = (56 \cdot 3) \text{ bits} = 168 \text{ bits}$
AES	$n = 128 \text{ bits}$	$k = 128, 192, 256 \text{ bits}$

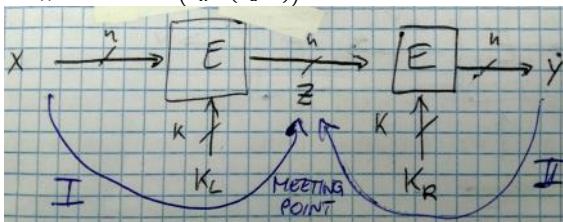
## DES

$n = 64 \text{ bits}$  |  $k = 56 \text{ bits}$

- 2DES
  - Encrypts twice:  $c = E(K_R, E(K_L, m))$
  - Meet-in-the-middle attack
    - Given  $x$  (input) and  $y$  (output)
    - An attacker could store all possible  $z_{L,i} = E(K_{L,i}, x)$  (the **meeting-point**, on  $n$  bits, it's a cipher's output) in a sorted data structure.

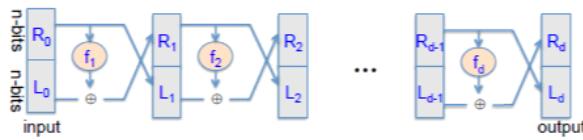
$k_{L1}$	$z_{L,1} = E(K_{L1}, x)$
$k_{L2}$	$z_{L,2} = E(K_{L2}, x)$
...	...
$k_{L,2^k}$	$z_{L,2^k} = E(K_{L,2^k}, x)$

    - Height:  $2^k$  entries
    - Width:  $k + n$  bits
    - Number of computation in order to build it:  $2^k$  (#entries) + [sorting operations]
    - The only advantage in respect to 1DES: very large and requires high number of computations
    - $z_{R,j} = D(K_{R,j}, y)$ , by decrypting the output  $y$  by means of all possible right-keys.
    - Search for  $z_{R,j}$  in the table containing all  $z_{L,i}$ 
      - Requires  $2^k$  computations
    - Overall cost:  $2^k + 2^k = 2^{k+1} = O(2^k) \Rightarrow$  same as 1DES
- 3DES
  - Encrypts-decrypts-encrypts:  $c = E(e_1, D(e_2, E(e_3, m)))$ 
    - Backward compatibility with 1DES if  $e_1 = e_2 = e_3$
  - Meet-in-the-middle attack requires  $O(2^{2k})$  computations, as if the key doubles
    - It's a meet-in-the-middle attack in 2DES, plus the last block cipher.
- $|K| < |P| = 56 < 64 \Rightarrow$  DES
  - So  $2^{56}$  permutations performed out of the total  $2^{64}$  possible ones: so DES isn't a perfect cipher.
  - The key is divided into multiple smaller keys, used in different rounds (48 rounds in 3DES)
- Based on the Feistel Network



Given functions  $f_1, \dots, f_d: \{0,1\}^n \rightarrow \{0,1\}^n$

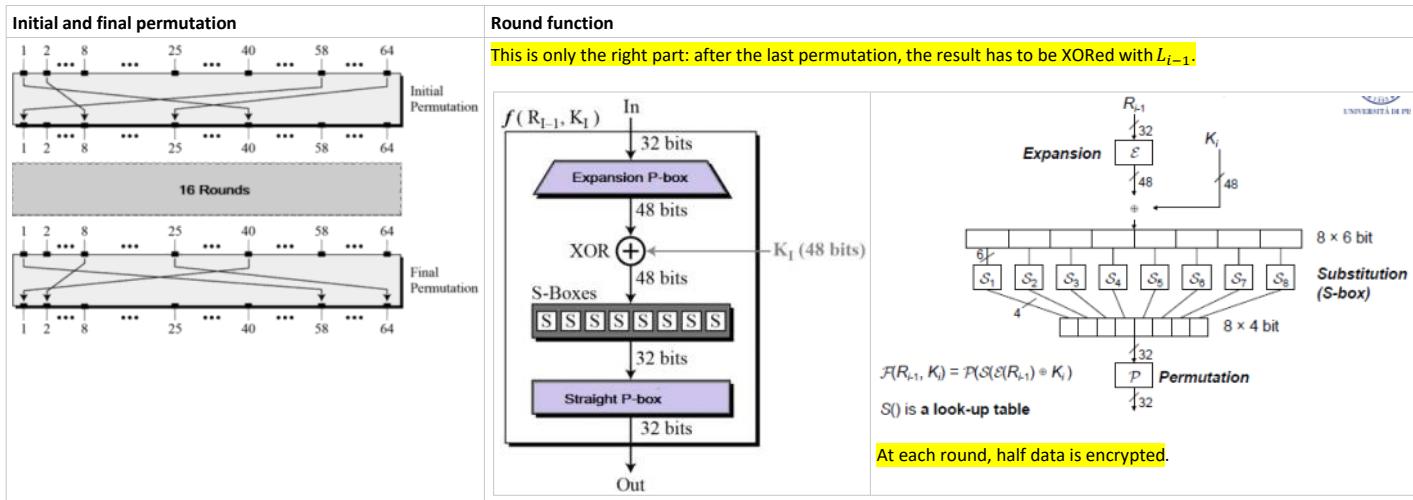
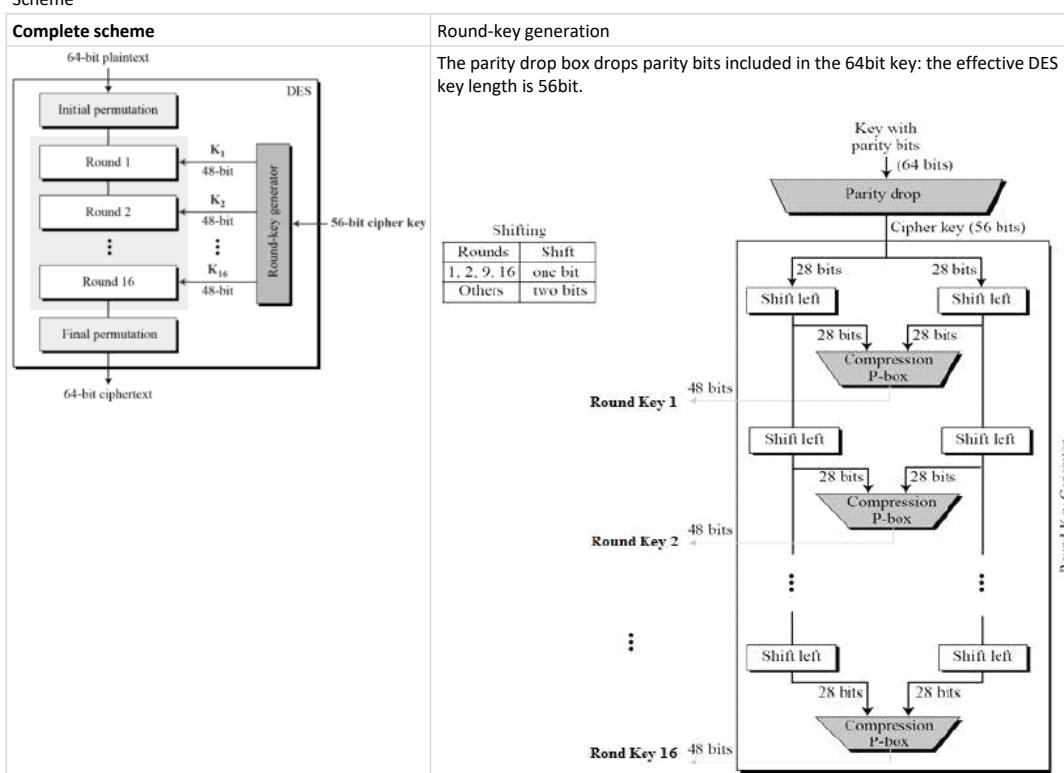
Goal: build invertible function  $F: \{0,1\}^{2n} \rightarrow \{0,1\}^{2n}$

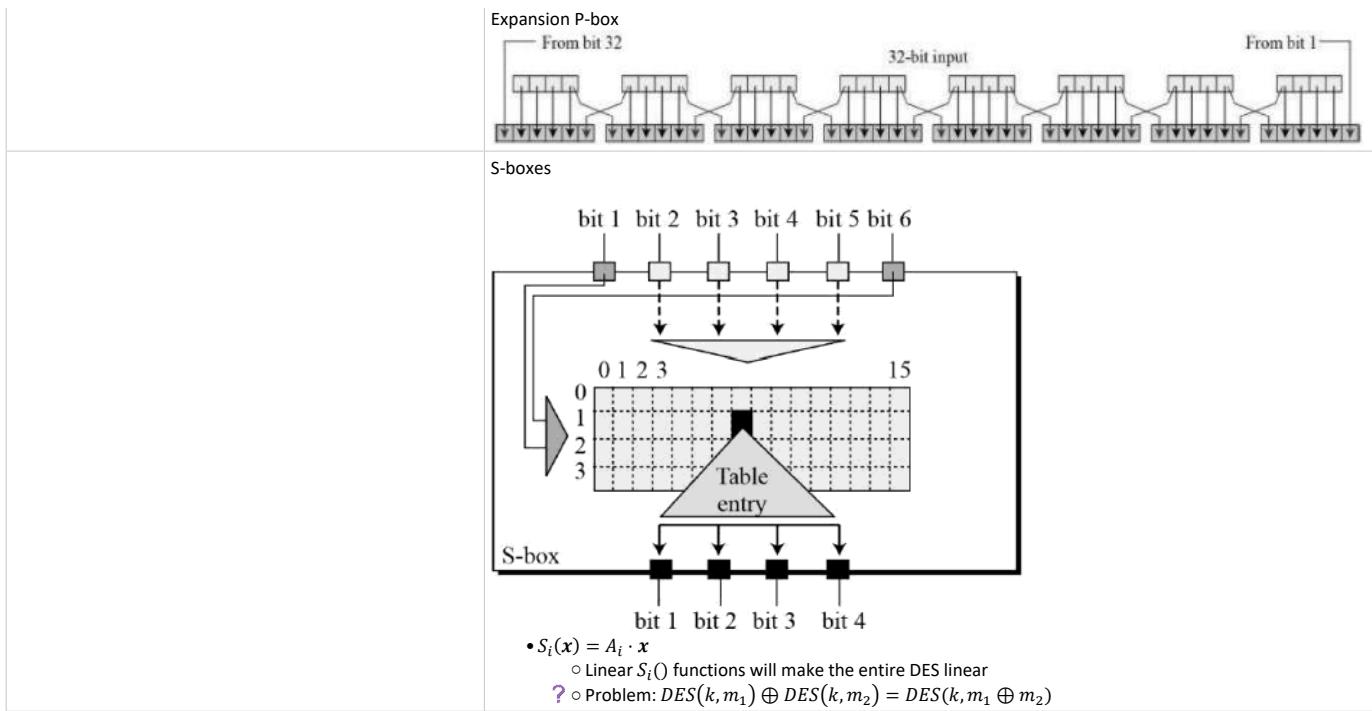


- o  $\begin{cases} L_i = R_{i-1} \\ R_i = L_{i-1} \oplus f_i(R_{i-1}) \end{cases} \Rightarrow \begin{cases} L_i = R_{i-1} \\ R_i = L_{i-1} \oplus f_i(L_i) \end{cases} = \begin{cases} R_{i-1} = L_i \\ L_{i-1} = R_i \oplus f_i(L_i) \end{cases}$
- Decryption uses the same circuit, with function applied in reverse order.
- DES actually uses  $R_i = L_{i-1} \oplus f_i(R_{i-1}, K_i)$
- o The Luby-Rackoff '85 version uses always the same function, with different keys applied to it. These function are PRFs.

- DES is a 16 round Feistel net

- o Legenda:
  - P-box: permutation box (which by itself it's the straight one)
    - Expansion P-box
    - Compression P-box
    - Straight P-box
  - S-box: substitution box
- o Scheme



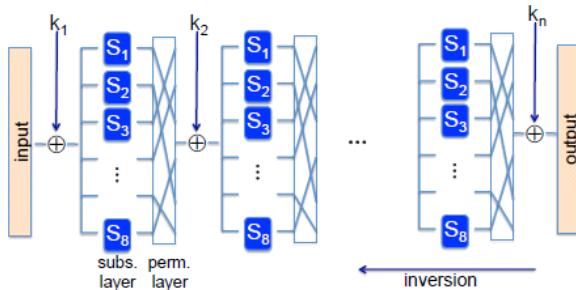


- Attacks
  - Exhaustive search: key is only on 56 bits
  - Chosen-plaintext attack

## AES

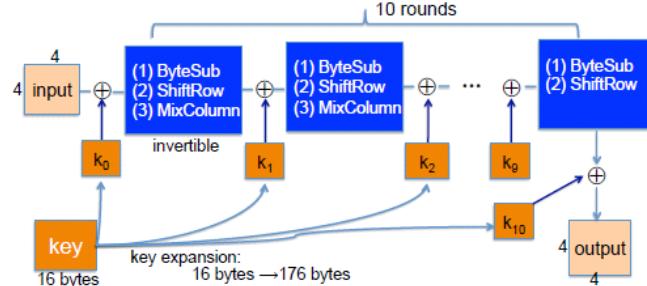
$n = 128 \text{ bits}$   $k = 128, 192, 256 \text{ bits}$

- 10, 12 or 14 rounds depending on the key size
  - At each round, all data is encrypted, therefore AES provides more security than DES even with less rounds. For this reason, AES also performs better.
- Based on Subs-Perms networks (encrypting/decrypting scheme)

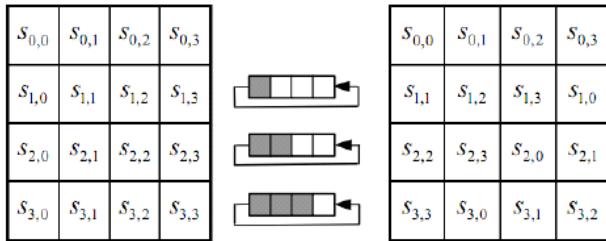


- Encryption is performed
- Inversion = decryption

- Actual AES scheme



- The input 128 bits long data block, is considered as a matrix of 4B by 4B
  - AES is Byte-oriented
- Round operations
  1. Byte substitution: implemented by a lookup table.
    - (256 1B entries)
    - Provides **confusion**
      - ◆ Relation b/w key and ciphertext is obscured
    - Called S-box (substitution-box)
    - Non-linear
    - Obviously invertible, because the decryption is done by inverting this operation
  2. ShiftRows
    - How is it done



- Provides **diffusion**

3. MixColumns: linear transformations (matrix multiplication made with another matrix with certain coefficients).

- o Round operations optimization

- Those operations are very simple, so they can be implemented via software.

To make it more performant, the first round can be implemented by a **huge** pre-computed lookup table that maps all different inputs to all different outputs.

	Code size	Performance
Pre-compute round functions (24KB or 4 KB)	Largest	Fastest (table lookups and xors)
Pre-compute S-box only (256 bytes)	Smaller	Slower
No pre-computation	Smallest	Slowest

- Javascript AES pre-computes it on client's browser upon receiving the AES library from a server.

- o In hardware: Intel has some instructions in its ISA to encrypt with AES

- Attacks

- o Related key attack in AES-256

- Given  $2^{99}$  ( $pt, ct$ ) pairs from four related keys, they can be recovered in  $2^{99}$ .

# Secret sharing

venerdì 09 giugno 2017 15:52

- 1 dealer
- $n$  players
- The dealer gives a share of the secret to the players, but only when specific conditions are fulfilled will the players be able to reconstruct the secret from their shares. The dealer accomplishes this by giving each player a share in such a way that any group of  **$t$  (for threshold)** or more players can together reconstruct the secret but no group of fewer than  $t$  players can.

## Trivial secret sharing

- **$t = 1$**

$t = 1$  secret sharing is very trivial. The secret can simply be distributed to all  $n$  participants.

- **$t = n$**

There are several  $(t, n)$  secret sharing schemes for  $t = n$ , when all shares are necessary to recover the secret:

I a. Encode the secret as an arbitrary length [binary](#) number  $s$ .

Give to each player  $i$  (except one) a random number  $p_i$  with the same length as  $s$ .

Give to the last player the result of  $(s \text{ XOR } p_1 \text{ XOR } p_2 \text{ XOR } \dots \text{ XOR } p_{n-1})$ .

The secret is the bitwise XOR of **all** the players' numbers ( $p$ ).

b. Additionally, (1) can be performed using any linear operator in any [field](#). For example, here's an alternative that is functionally equivalent to (1). Let's select 32-bit integers with well-defined overflow semantics (i.e. the correct answer is preserved, modulo  $2^{32}$ ). First,  $s$  can be divided into a vector of  $M$  32-bit integers called  $v_{\text{secret}}$ . Then  $(n-1)$  players are each given a vector of  $M$  random integers, player  $i$  receiving  $v_i$ . The remaining player is given  $v_n = (v_{\text{secret}} - v_1 - v_2 - \dots - v_{n-1})$ . The secret vector can then be recovered by summing across all the player's vectors.

- **$1 < t < n$ , and, more general, any desired subset of  $n$**

The difficulty lies in creating schemes that are still secure, but do not require all  $n$  shares.

For example, imagine that the Board of Directors of a company would like to protect their secret formula. The president of the company should be able to access the formula when needed, but in an emergency any 3 of the 12 board members would be able to unlock the secret formula together. This can be accomplished by a secret sharing scheme with  $t = 3$  and  $n = 15$ , where 3 shares are given to the president, and 1 is given to each board member.

When space efficiency is not a concern, trivial  $t = n$  schemes can be used to reveal a secret to any desired subsets of the players simply by applying the scheme for each subset. For example, to reveal a secret  $s$  to any two of the three players Alice, Bob and Carol, create three different (2,2) secret shares for  $s$ , giving the three sets of two shares to Alice and Bob, Alice and Carol, and Bob and Carol.

# Hash functions

martedì 23 maggio 2017 11:28

## ★ • Integrity

- Properties
  - Compression
  - Computationally easy
- Definition:  $H: \{0,1\}^* \rightarrow \{0,1\}^m$ 
  - So  $m$  is the digest length
  - $n$  usually is the input length (could be any size)
- Hash functions

	<b><math>m</math></b>	<b>Preimage</b>	<b>Collision</b>
MD5	128	$2^{128}$	$2^{64}$
SHA-1	160	$2^{160}$	$2^{80}$
SHA-256	256	$2^{128}$	$2^{128}$
SHA-512	512		$2^{256}$

- Security properties

	<b>Breaking complexity</b>	<b>One-way hash functions (OWHF) or weak one-way hash function</b>	<b>Collision resistant hash functions (CRHF) or strong one-way hash function</b>
<b>1. Preimage resistance</b> <i>finding <math>x</math> given <math>h(x)</math></i>		Yes	Almost always in practice
<b>2. 2nd-preimage resistance</b> <i>given <math>x</math>, finding <math>x' \neq x</math> such that <math>h(x) = h(x')</math></i>	Black box attack complexity $O(2^m)$	Yes	Yes (implied)
			
<b>3. Collision resistance</b> <i>finding <math>(x, x')</math> such that <math>h(x) = h(x')</math></i>	Birthday attack complexity $O(2^{(m/2)})$	No	Yes

- Black box attacks

The hash function  $H$  is considered as a black box which produces a  $m$ -bit long output considered as a random variable.

- Guessing attack

<b>Complexity</b>	$O(2^m)$
<b>Which security property is attacked</b>	<b>2. 2nd-preimage resistance</b> <i>given <math>x</math>, finding <math>x' \neq x</math> such that <math>h(x) = h(x')</math></i>
<b>Algorithm</b>	repeat $x \leftarrow \text{random}()$ until $h(x_0) = h(x)$ return $x$
<b>Storage complexity</b>	Costant

- Bithday attack

<b>Complexity</b>	$O(2^{m/2})$
<b>Which security property is attacked</b>	<b>3. Collision resistance</b> <i>finding <math>(x, x')</math> such that <math>h(x) = h(x')</math></i>
<b>Birthday paradox theorem</b>	• $r_1, r_2, \dots, r_n \in \{1, \dots, B\}$ ( $B = 2^n$ ), independent and distributed integers • When $n = 1.2 \cdot B^{1/2} \Rightarrow \Pr\{\exists i, j, i \neq j: r_i = r_j\} \geq \frac{1}{2}$ (collision probability)
<b>Algorithm</b>	1. Choose $N = 2^{n/2}$ random input messages $x_1, \dots, x_N$ 2. For $i = 1$ to $N$ , compute $H(x_i)$ 3. Look for a collision. If not found, go to step 1.
<b>Storage complexity</b>	$2^{n/2}$

- Integrity schemas with Collision Resistant Hash Functions (CRHFs)

$m \parallel E(e, H(m))$

- ?
- Sender has seen  $H(m)$   
(as seen on the slides)
- No confidentiality  
Message  $m$  sent in clear
- The receiver cannot be sure that the  $m$  comes from the actual sender.  
The attacker could have found a 2nd-preimage that passes the hash check.

$E(e, m) \parallel H(m)$

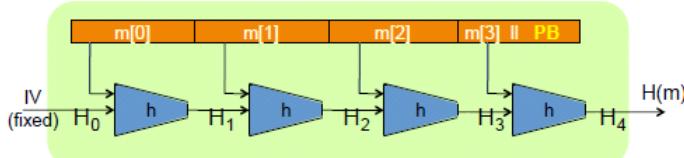
- $H(m)$  can be used to check a guessed  $m$

$E(e, m \parallel H(m))$

- Confidentiality  
No plaintext ( $m$ ) sent over the network
- Integrity  
Hash sent with the message
- As secure as  $E$   
If  $E$  fails, everything fail

- The Merkle-Damgard iterated construction

- It is a CRHF for short message that is able to hash long messages too
  - Scheme



- If  $h$  is collision resistance, then so is  $H$ .
  - The compression function  $h$  can be a **cipher**
    - Key: IV or previous compression output
    - Pros
      - Reduces the size of SW libraries
      - Reduces the size of HW circuits

- MD5 and SHA-1 are not collision resistance
  - They're still useful when  $x$  is given, because MD5 and SHA-1 are 2nd-preimage resistance.

# MACs

martedì 23 maggio 2017 11:26

## ★ • Authentication

- Stands for : "Message Authentication Code"
- Its output is called **tag**
- Properties
  - Compression
  - Computationally easy
  - Computation-resistance: it should be impossible to compute a tag without knowing the key.

- Practical HMACs

<b>SSL</b>	$c = E(e, m \parallel S(a, m))$
<b>IPsec</b>	$c \parallel t = E(e, m) \parallel S(a, E(e, m))$
<b>SSH</b>	$c \parallel t = E(e, m) \parallel S(a, m)$

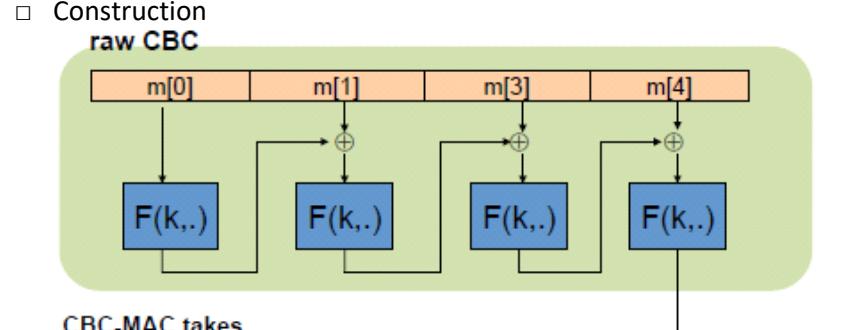
- Theorem:

If:

- $F: K \times X \rightarrow Y$  (MAC generation function) is a secure PRF
- $1/|Y|$  is negligible ( $|Y| \geq 2^{80}$ )

Then  $F$  defines a secure MAC

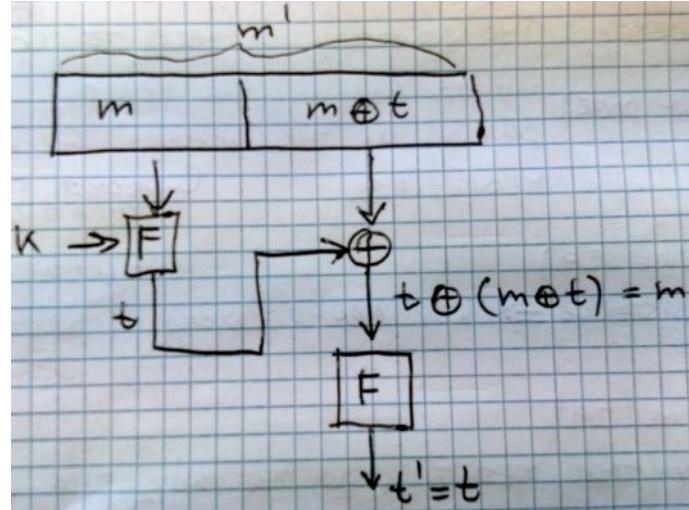
- Problem: AES' input is 128 bit long. Excluding the key bits, this is a MAC for small messages.  
To convert small-MACs into large ones, CBC-MAC and HMAC are used.
- How they're made
  - From PRFs (*Pseudo-Random Functions*, like ciphers)
    - CBC-MAC



CBC-MAC takes

- two independent keys  $K$  and  $K_1$
- an arbitrary # of input blocks
- PRF is a cipher

- ♦ Existential forgery w/o last encryption



- Maximum number of messages authenticated with CBC-MAC with the same key
  - ◆ After  $q$  messages, CBC-MAC becomes insecure with a probability of  $P = q^2/|X|$ 
    - ◊  $q = \#$  messages CBC-MAC<sup>†</sup> ed with the same key
    - ◊  $|X| = \text{input block size}$
  - ◆ Fixed  $P = 1/2^{32}$ 
    - ◊ AES:  $|X| = 2^{128} \Rightarrow q < 2^{48}$
    - ◊ 3DES:  $|X| = 2^{64} \Rightarrow q < 2^{16}$

? ■ CMAC (*Cipher-based MAC*)

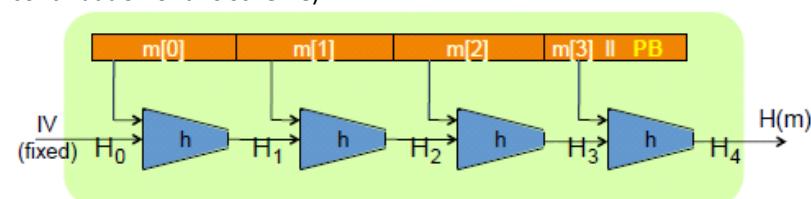
- From CRHFs (*Collision Resistance Hash Functions*)
  - HMAC (*Hash-based MAC*)

- Insecure scheme

- ◆  $S(k, m) = H(k \parallel m)$
  - ◆ Suffers from the existential forgery attack.

Starting from a  $[m, S(k, m)]$  pair, the adversary can compute the  $[m \parallel \text{padding} \parallel w, h(w, t)]$  pair, where  $w$  is a block.

Proof is obvious just by looking at the Merkle-Damgård hash construction (it's just a continuation of this scheme):



- Standard

- ◆  $S(k, m) = H((k \oplus opad) \parallel H((k \oplus ipad) \parallel m))$ 
    - ◊ This first hash  $H((k \oplus ipad) \parallel m)$  takes time because its input is large.
    - ◊ The second hash then takes less time.
    - ◊ *opad* and *ipad*: fixed and predefined
    - ◊ SHA-1 is not collision resistant, but HMAC only needs the compression property

- Timing attack: some verifiers returned false as soon as the first computed tag byte was different from the received one.

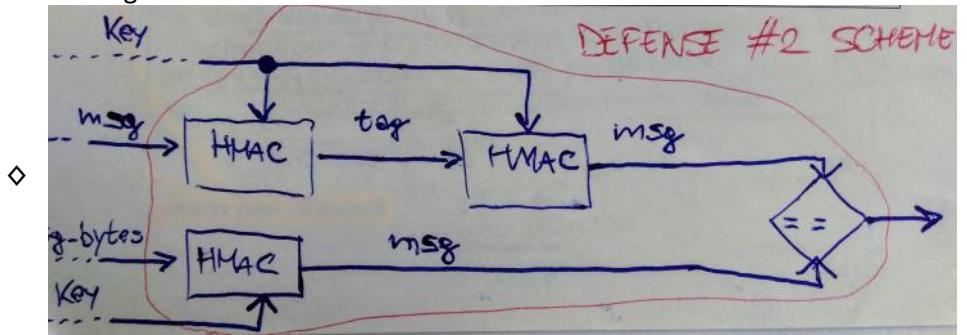
- ◆ Defense #1

- ◊ Make string comparator **always take same time**.
    - ◊ Can be difficult to ensure due to **optimizing compiler**.

- ◆ Defense #2

- ◊ By HMAC-ing the tag, the receiver gets the message.
    - ◊ Instead of comparing it to the message itself, it compares it with the

message hashed two times



- Padding
  - By zeros
    - $\text{pad}(m)$  and  $\text{pad}(m||0)$  produce the same MAC
  - Standard padding (ISO)
    - "100...00", scan from the right
    - Dummy block to avoid existential forgery

# Diffie-Hellman

martedì 23 maggio 2017 11:17

- ★ • Symmetric shared key establishment

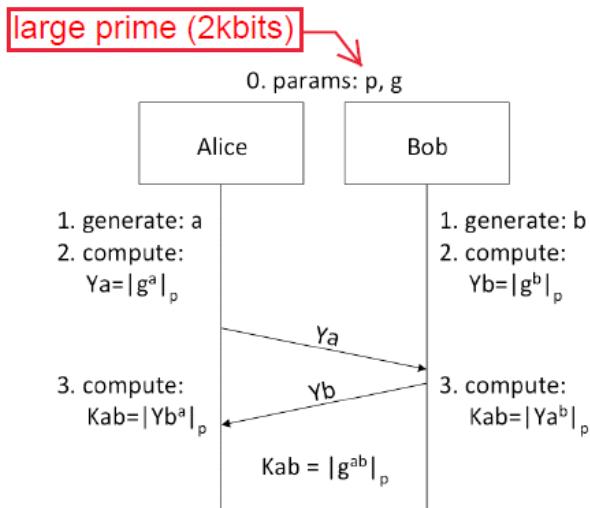
- Key management strategies

	<b>Pairwise keys</b>	<b>TTP (Trusted Third Party)</b>
<b>Pros</b>	• Only one communication at the time can be compromised	• High scalability <ul style="list-style-type: none"> <li>• <math>n</math> keys</li> <li>• Easy to add/remove entities</li> </ul>
<b>Cons</b>	• Poor scalability <ul style="list-style-type: none"> <li>• <math>O(n^2)</math> keys</li> <li>• Hard to add/remove entities</li> </ul>	• TTP single point of failure <ul style="list-style-type: none"> <li>• All communications can be compromised</li> <li>• TTP always online</li> <li>• TTP knows all keys</li> </ul>

- Math
  - Theorem:  $\forall 1 \leq t < p (\mathbb{Z}_p), \exists x \text{ s.t. } g^x \bmod p = t, p \text{ prime and } g \in (1, p)$
  - **Discrete exponentiation:** given  $g, p$  and  $x$ , it's **easy to compute**  $y = g^x \bmod p$ 
    - Square-and-multiply algorithm:  $2^{20} = (2^{10})^2 = ((2^5)^2)^2 = ((2 \cdot 2^4)^2)^2 = ((2 \cdot (2^2)^2)^2)^2$
    - Discrete ( $\bmod p, |p| = n$ ) exponentiation (square-and-multiply algorithm) takes at most  $2 \cdot \log_2 p \leq 2 \cdot \log_2 2^n = 2n$  multiplications
  - **Discrete logarithm:** given  $g, y$  and  $p (1 \leq y \leq p - 1)$ , it's **difficult to find**  $x$  s.t.  $y = g^x \bmod p$ 
    - Discrete ( $\bmod p, |p| = n$ ) logarithms takes at most  $p^{1/2} \leq 2^{n/2}$  operations (way more harder)

## • Diffie-Hellman

- Sharing keys w/o a TTP
- Scheme



- Secure against **eavesdropping** ([DH problem](#) = discrete logarithm problem) (passive adversary)
  - What the adversary sees
    - $p$  and  $g$  (can be standard quantities or exchanged between A and B)
    - $Y_a$  and  $Y_b$  (eavesdropped)
  - In order for the adversary to compute  $K_{ab} = g^{ab} \bmod p$ , s/he has to discover at least

*a or b.*

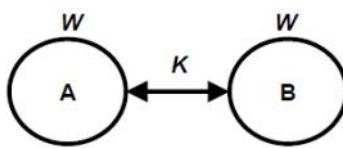
□ This is the [discrete logarithm problem](#) for *a* or *b*. Recall:  $2^{n/2}$  operations.

💬 o Insicure against a **MITM** (active adversary)

- $g$  is public and known, so the adversary can send  $Y_c = g^c \bmod p$  to the other party
- At the end of the attack, A and B won't have equal keys
  - A:  $g^{ac} \bmod p$
  - B:  $g^{bc} \bmod p$
- Problem:  $Y_a$  can be thought as A's public key, so there's nothing that links  $Y_a$  with A  $\Rightarrow$  certificates.

• Session or Ephemeral keys

### Session/Ephemeral key



- A and B a priori share a *long term key*  $W$
- A and B wants to establish a *session key*  $K$

- Session key is used for bulk encryption
- A session key is used for one communication session
- Long term key is used for many runs of the key establishment protocols; in each run, the key encrypts a small amount of data

UNIVERSITÀ DI PISA

**Establishing an ephemeral/session key**

**one-pass**

M1  $A \rightarrow B: E(W, t_A \parallel "B,A" \parallel K)$

**with challenge-response**

M1  $A \leftarrow B: n_B$

M2  $A \rightarrow B: E_W(W, n_B \parallel "A,B" \parallel K)$

**both parties contribute to the session key**

M1  $A \leftarrow B: n_B$

M2  $A \rightarrow B: E(W, K_A \parallel n_B \parallel n_A \parallel "A,B")$

M3  $A \leftarrow B: E(W, K_B \parallel n_A \parallel n_B \parallel "B,A")$   
A.A. 2012-2013 Key management

•  $t_A$  is a timestamp (a "fresh" quantity) requires synchronized clocks

•  $n_B$  is a nonce (a "fresh" quantity)

•  $n_A$  and  $n_B$  are nonces

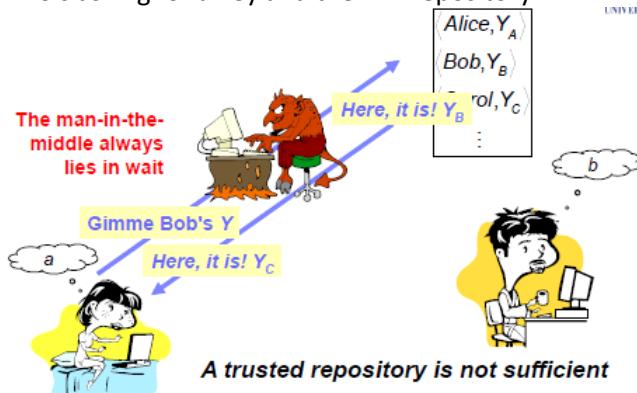
•  $K_A$  and  $K_B$  are keying materials

•  $K = K_A \circ K_B$

# Public key encryption

domenica 04 giugno 2017 17:35

- Consists in a triple of algorithms
  - G: key generation, produces (*public key, secret key*)
  - E: encryption (with destination's public key)
  - D: decryption (with personal private key)
- Not perfect for Shannon
  - Adversary intercepts a ct
  - Adversary selects a pt  $m$  s.t.  $\Pr[M = m] \neq 0$
  - Adversary computes  $c' = E(pk, m)$ . If  $c \neq c' \Rightarrow \Pr[M = m | C = c] = 0 \neq \Pr[M = m]$
- Insicure against a **MITM**
  - If the sender sends its name and public key at first, the MITM can change the latter.
  - A trusted public key repository is not enough: an adversary can perform a MITM attack between someone who's asking for a key and the TTP repository.



That's why there exists:

- Key distribution **with public keys**
  - A TTP (Certification Authority) certifies the public key of each entity.
  - Certificate:

$$< Bob, PK_{Bob}, SK_{TTP}(Bob \parallel PK_{Bob}) >$$

A user, in order to get Bob's public key, has to decrypt the last part of his certificate with the CA's public key.

- Comparison among crypto-systems

	Elliptic curves	Diffie-Hellman	RSA
Private key operations	Faster	Slower	Slower
Public key operations	Slower	Slower	Faster if $e$ is small

# RSA

lunedì 05 giugno 2017 10:11

## ★ • Generates (*public key, secret key*) pairs

- Consists in two algorithms
  - ★ ○ Provides an **encryption and decryption** scheme
  - ★ ○ **Digital signatures**

- Strictly related to the factoring problem

- Lengths
  - $|n| = |e| = |d| = k = \log_2 n + 1 = \log_2 e$
  - $|p| = |q| = \frac{|n|}{2}$ 
    - Because  $n = p \cdot q$

- **Key generation** algorithm with large prime numbers
  - Algorithm

1. Generate two large, distinct **primes  $p$  and  $q$**  (100÷200 digits)
2. Compute  $n = p \cdot q$  and  $\phi(n) = (p - 1) \cdot (q - 1) = \varphi(n)$
3. Select a random number (public key)  $1 < e < \phi(n)$  such that  $\gcd(e, \phi(n)) = 1$ 
  - i)  $e$  exists
4. Compute the **unique** integer (private key)  $1 < d < \phi$  such that  $d \cdot e = 1 \pmod{\phi}$ 
  - i) There is an integer  $k$  s.t.  $d \cdot e = k \cdot \phi$
  - ii) Can be done efficiently by means of the *Extended Euclidean Algorithm* (EEA) in a logarithmic time
5. Generated keys:
  - **Public key:**  $(n, e)$ 
    - $e$ : encryption key
    - $n$ : maximum message numerical value
  - **Private key:**  $(n, d)$ 
    - $d$ : decryption key
    - $n$ : maximum message numerical value
6. [Destroy  $p$  and  $q$ ]  
(Cannot be done if decryption needs optimization on low performance devices: see [1](#) and [2](#))

- How to find large prime numbers (for  $p$  and  $q$ )

- Algorithm

```
repeat
    p ← randomOdd(x);
until isPrime(p);
```

- On average  **$\ln(x)/2$  odd numbers must be tested** before a prime  $p < x$  can be found

Proof:

- In a  $[0, x]$  interval there exist  $x/\ln(x)$  prime numbers on average.

- Probab. that a random odd number is prime  $= \frac{x/\ln(x)}{x/2 \text{ (#odd numbers)}}$  $= \frac{2}{\ln(x)}$

- Primality tests do not try to factor the tested number

- **Encryption:**  $c = |m^e|_n$ , once  $m$  has been represented as an integer **belonging to the interval  $[0, n - 1]$**

- How efficient can the exponentiation operation can be done

- Grade-school algorithm requires  $(d - 1)$  multiplications
  - If  $d$  is large as  $n$ , this algorithm is inefficient
- **Square-and-multiply algorithm** requires up to  $2k$  multiplications ( $\text{modulus } n \in \{0,1\}^k$ ): since a multiplication of two  $k$ -bits numbers can be done in  $O(k^2)$ , the whole exponentiation operation can be done in  $O(2k \cdot k^2) = O(2k^3) = O(k^3)$ .

The next point will explain why:

- $|m^e|_n$  requires at most  $\log_2(e)$  multiplications and  $\log_2(e)$  squares.  
In this case,  $k = \log_2(e)$  is the number of bits of the exponent  $e$ .

Let  $e_{k-1}, e_{k-2}, \dots, e_2, e_1, e_0$ , where  $k = \log_2 e$ , the binary representation of  $e$

So  $e_i = \{0, 1\} \forall i \in [0, k - 1]$ , are bits, zeros or ones.

All the following numbers ( $m$  and the entire exponent) are expressed in base 10:

$$\begin{aligned} m^e \bmod n &= m^{(e_{k-1}2^{k-1} + e_{k-2}2^{k-2} + \dots + e_22^2 + e_12 + e_0)} \bmod n \equiv \\ &m^{e_{k-1}2^{k-1}} m^{e_{k-2}2^{k-2}} \cdots m^{e_22^2} m^{e_12} m^{e_0} \bmod n \equiv \\ &\left( m^{e_{k-1}2^{k-2}} m^{e_{k-2}2^{k-3}} \cdots m^{e_22} m^{e_1} \right)^2 m^{e_0} \bmod n \equiv \\ &\left( \left( m^{e_{k-1}2^{k-3}} m^{e_{k-2}2^{k-4}} \cdots m^{e_2} \right)^2 m^{e_1} \right)^2 m^{e_0} \bmod n \equiv \\ &\left( \left( \left( m^{e_{k-1}} \right)^2 m^{e_{k-2}} \right)^2 \cdots m^{e_2} \right)^2 m^{e_1} \right)^2 m^{e_0} \bmod n \end{aligned}$$

◆ Algorithm:

```
c = 1; // final result
for(i = k-1; i >= 0; i--) { // starting from the MSB
    c = c2 mod n; // the exponentiation is made before the
                      // multiplication. At first, 12 is computed.
                      // requires k squares
    if(ei == 1) // otherwise it's 0, then m0 = 1
        c = (c * m) mod n; // starting from the left, all
                            // multiplications are performed, only
                            // when ei ≠ 1 so mei won't be 1.
                            // as many multiplications as the
                            // number of ei = 1 (at most k).
}
return c;
```

Up to  $2k$  multiplications. Every multiplication requires  $O(k^2)$ , so the total computational cost is  $O(2k \cdot k^2) = O(2k^3) = O(k^3)$

○ How to select  $e$

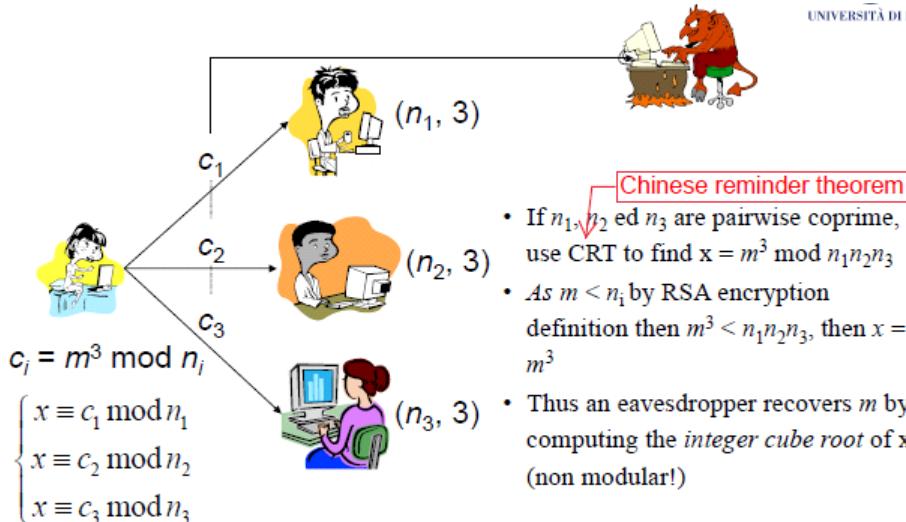
- Could be **random** (by definition)
- **Public** (belongs to the public key)
- **Co-prime with  $\phi(n)$**  (such that  $\gcd(e, \phi(n)) = 1$ ).
- The number of square-and-multiply modular multiplications is equal to the number of 1s in  $e$ , so particular values of  $e$  **containing a small number of 1s** require a smaller

number of modular multiplications in the algorithm (the number of squares cannot be reduced).

e	Binary representation	# modular multiplications
3	11	2
17	1 0001	5
$2^{16} + 1$	1 0000 0000 0000 0001	17

- Why  $e = 3$  is never used due to the [low exponent attack](#)
- Why one "1-bit" isn't enough:
  - ◆ If the LSB is equal to 1, it means that  $e = 1$ , hence, no encryption is provided.
  - ◆ In all the other cases, the LSB is set to zero, so  $e$  is even and it cannot be co-prime with  $\phi(n)$ .

- **Decryption:**  $m = |c^d|_n$ 
  - How to select **d**
    - Must satisfy Diffie-Hellman
    - In order to discourage brute-force attacks:
      - The number of 1s cannot be pre-set, otherwise the key space would reduce itself.
      - It should be sufficiently large, as close as  $n$  as possible.
- Security: the RSA Problem (**RSAP**)
  - Recovering plaintext  $m$  from ciphertext  $c$ , given the public key  $(n, e)$
  - As long as this problem is hard, RSA is hard
  - Methods
    - Computing the decryption exponent **d** from the public key  $(n, e)$  to decrypt with  $m = |c^d|_n$ 
      - It is equivalent to **factoring  $n$** .
        - ◆ By "factoring  $n$ ", the adversary finds out  $p, q$ , and so  $n = p \cdot q$ ,  $\varphi(n) = (p - 1) \cdot (q - 1)$  and finally  $d$  thanks to  $d \cdot e = 1 \pmod{\varphi}$ , like in the [key generation algorithm](#).
        - ◆ Exploits the  $RSAP \leq_P FACTORING$  property: RSAP is not harder than FACTORING: if the attacker is able to factorize  $n$ , then s/he is able to solve the RSAP.
      - Knowing  $\varphi(n)$  is computationally equivalent to factoring  $n$ .
    - Proof:
      - ◆ If  $p$  and  $q$  are known s.t.  $n = p \cdot q \Rightarrow \varphi(n) = (p - 1) \cdot (q - 1)$
      - ◆ If  $\varphi(n)$  is given (also  $n$  is known, it's part of the public key):
        - a. From  $\varphi = (p-1)(q-1) = n - (p+q) + 1$ , determine  $x_1 = (p + q)$ .
        - b. From  $(p - q)^2 = (p + q)^2 - 4n = x_1^2 - 4n$ , determine  $x_2 = (p - q)$ .
        - c. Finally,  $p = (x_1 + x_2)/2$  and  $q = (x_1 - x_2)/2$ .
    - $e$ -th root of  $c = |m^e|_n$ 
      - Computing it is computationally easy iff  $n$  is prime
      - If  $n$  is composite this computation is equivalent to factoring
    - $p$  and  $q$  should be large and about the same bitlength in order to make factoring  $n = p \cdot q$  computationally infeasible (to avoid the elliptic curve factoring)
  - **Drawbacks/attacks** and solutions
    - Low exponent attack



- If the **message space is small**, the adversary can decrypt and create new messages. This can happen with bits or, in general, with a limited set of numbers (**auctions**, for example): the adversary can discover Alice's bid by crypting all possible numbers with her cipher and confronting them with the actual ciphertext that Alice sent (her bid).

- **Solution:** make the message artificially longer (it's called "**salting the message**") by choosing a random quantity  $r$  and then sending  $c = |(r \parallel \text{bid})^e|_n$

- **Malleability:** RSA satisfies the **Homomorphism property**:  $(m_1 \cdot m_2)^e = m_1^e \cdot m_2^e = c_1 \cdot c_2 \text{ mod } n$ .

This property can cause **messages alteration** (malleability) because an adversary can introduce a controlled modification of the plaintext as follows:

- The adversary has a ciphertext  $y = |x^e|_n$
- By selecting a number  $s$  such that  $\gcd(s, n) = 1$ , the adversary can compute  $y' = |y \cdot s^e|_n$
- The receiving side decrypts:  $x' = |(y')^d|_n = |(y \cdot s^e)^d|_n = |(x^e \cdot s^e)^d|_n = |x \cdot s|_n$

This can be a problem when  $x$  represents a number, for example.

- **Solution:** it consists in introducing **redundancy**, by duplicating the message as follows:

$$y = |(x \parallel x)^e|_n$$

If the adversary performs the same tricks, after decryption, the receiving side gets  $x' = s(x \parallel x)$ : the receiver can then search for repeated bits: it's very unlikely that the adversary finds such an  $s$  that can produce a perfectly duplicated bit flow at the receiving side. There's no limit in redundancy.

In both cases, message tags (HMACs) can be useful for redundancy, plus, the salting could also consist in a random bit configuration.

The **PKCS#1 padding standard** tells how to do this in order to avoid those kind of attacks.

# ElGamal

lunedì 05 giugno 2017 10:11

- Based on the [Discrete Logarithm Problem](#) (DLP)
- **Encryption:**
  - Select  $k$  randomly
  - Send:  $|g^k|_p \parallel |m \cdot y^k|_p$ 
    - Where  $g$  is the generator (shared secret)
    - $y$  is the public key
- **Decryption:**
  - $c_1^x = |g^{kx}|_p = |y^k|_p$ 
    - $g^x = y$  by definition
  - $m = |m \cdot y^k|_p \cdot |y^{-k}|_p$
- **Security**
  - a.  $k$  is random  $\Rightarrow |g^k|_p, |y^k|_p$  are random.
  - b. An adversary needs  $|y^k|_p$  to decrypt.
  - c. The task of calculating  $|y^k|_p$  from  $(g, p, q)$  is equivalent to the [Diffie-Hellman Problem](#) (DHP), and thus based on the [Discrete Logarithm Problem](#) (DLP) in  $\mathbb{Z}_p$ .
  - d.  $k$  must be different everytime, otherwise the key can be found.

# Elliptic curve cryptography

lunedì 05 giugno 2017 10:12

- Based on the Elliptic Curve Discrete Logarithm Problem (**ECDLP**).
  - There's no easy way to find  $k$  that solves  $\mathbf{Q} = k \cdot \mathbf{G}$  (multiplication is redefined) on curves, where:
    - $\mathbf{Q}$  is the public key
    - $k$  is the private key
    - $\mathbf{G}$  is the generator
  - ECC **parameters** consists in the parameters of the elliptic curve ( $y^2 = |x^3 + ax + b|_p$ ).
- Re-definitions
  - Both Diffie-Hellman and ECC are defined on an additive [Abelian groups](#), so Diffie-Hellman can be redefined on this particular group, **where the exponentiations are substituted with the  $\mathbf{Q} = k \cdot \mathbf{G}$  operations (ECDH)**.  
This is possible because all the numbers smaller than  $p$ , with all the possible operations, form an additive Abelian group.
  - Also ElGamal has been redefined on elliptic curves, and it is called **ECDSA**.
- Pros and cons

Pro	Con
<ul style="list-style-type: none"><li>• <b>Keys are smaller than RSA ones</b> (by providing the same amount of security)</li></ul>	<ul style="list-style-type: none"><li>• If certain curves are selected, the ECDLP becomes easy to solve.</li></ul>

# Side channel attacks



martedì 23 maggio 2017 11:15

12.side\_cha  
nnels\_att...

- A side channel attack is based on information gained from the **physical** implementation of a cryptosystem
  - Strong magnetic fields attacks
  - Physical alteration
- The adversary
  - Has physical access to the device
  - Knows the used algorithm: the only secret is the key
- Three types:
  - Fault injectionRSA requires several exponentiations (thousands of multiplications on 1024 bits: each exponentiation requires about 1500 multiplications).
  - **Decryption optimizations** are needed for low performance devices:
    - Exponents with a small number of 1s
    - Prime exponents

Based on the [Chinese Remainder Theroem](#) (CRT):

a) **Compute**  $c_p = |c|_p$  and  $c_q = |c|_q$

In the original RSA,  $p$  and  $q$  had to be deleted at the end of the key generation algorithm: in order to apply this optimization,  $p$  and  $q$  cannot be deleted at the end of the key generation algorithm.

b) **Compute**  $m_p = \left| c_p^{\lfloor d \rfloor_{p-1}} \right|_p$  and  $m_q = \left| c_q^{\lfloor d \rfloor_{q-1}} \right|_q$

In order to calculate  $m_p$  and  $m_q$ , two exponentials have to be computed.

 One exponential operation requires  $1.5 \cdot t$  computations (on average, I guess): now the exponent  $\lfloor d \rfloor_{p-1}$  is on the number of bits of  $p$  and  $q$ , that have half number of bits of  $n$ .

◊ Before:  $|n| = |d| = t$ , previously  $k$

◊ Now:  $\lfloor d \rfloor_{p-1} = |p| = |q| = \frac{|n|}{2}$  (because  $n = p \cdot q$ )

So in this case 2 exponentiations on  $t/2$  bits are needed.

In total, the number of multiplications remains the same:  $2 \cdot \left( 1.5 \cdot \frac{t}{2} \right) = 1.5 \cdot t$  multiplications.

What is changed here is the operation complexity: instead of  $O(t^2)$ , here we have

$$O\left(\left(\frac{t}{2}\right)^2\right) = O(t^2/4).$$

## Summary

$m = \left| c^d \right|_n$  requires  $1.5 \cdot t$  multiplications on  $t$  bits, that have a complexity of  $O(t^2)$ .

With the CFT, still  $1.5 \cdot t$  multiplications are required, but the complexity drops down to  $O(t^2/4)$  (because those multiplications involve a smaller number of bits), with a performance speed-up of 4 (4 times quicker).

## Conclusion

Also low performance devices such as smartcards can compute RSA decryption.

c) **Compute**  $m = a_p m_p q + a_q m_q p$

$a_p$  and  $a_q$  can be pre-computed: another reason why  $p$  and  $q$  cannot be deleted at the end of the key generation algorithm.

- The adversary can make at least **one of the two computations of  $m$  fail**.

This can be done by putting the smartcard under a strong magnetic field.

For example, the adversary can change  $m_p$  to  $m'_p$ , thus,  $m$  changes into  $m'$ .

If the **computation is run twice**, both  $m$  and  $m'$  can be computed.

□  $m = a_p m_p q + a_q m_q p$

□  $m' = a_p m'_p q + a_q m_q p$

The highlighted terms cancel each other, so  $m - m' = a_p \cdot (m_p - m'_p) \cdot q$ , so the difference  $(m - m')$  is a multiple of  $q$ .

Also another quantity is a multiple of  $q$ , which is  $n = p \cdot q$ .

So  $q$  becomes the greatest common denominator between  $n$  and  $(m - m')$ :

$$\square \quad \gcd(n, m - m') = q$$

**Computing it** is computationally **easy** (with the EEA algorithm).

In this way, if the adversary can compromise the RSA key generation algorithm with a side channel attack, s/he can compute  $q$  quite easily.

Note that in this case, the implementation is attacked, not the algorithm.

- o Simple power analysis (SPA)

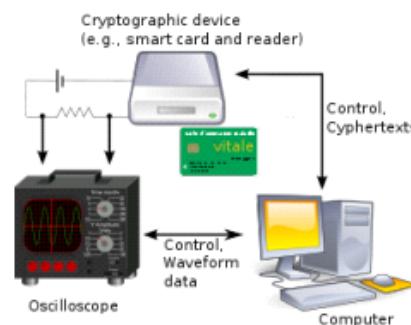
Consists in analysing the power consumption of the cryptographic hardware device: in this way, the attacker does not need any physical interaction with the device.

For example, in RSA, more power is needed during multiplications operations (when the key bit is 1) than the no multiplication period (key bit is 0).

- **Differential power analysis (DPA)**

involves statistically analyzing power consumption measurements from a cryptosystem.

- DPA attacks have signal processing and error correction properties which can extract secrets from measurements which contain too much noise to be analyzed using simple power analysis.



- o Time analysis

- For example the timing attack against HMAC, [as already shown during previous classes](#).

# Digital signatures

martedì 23 maggio 2017 11:14

## Difference between PKE and DS

Public Key Encryption	Digital Signatures
<ul style="list-style-type: none"><li>The sender encrypts the message <u>with the destination's public key</u>.</li><li>Only the destination can read the message, because its private key it's needed to decrypt it.</li><li>Mainly used for data confidentiality (<b>secrecy</b>).</li></ul>	<ul style="list-style-type: none"><li>The sender encrypts the message <u>with its own private key</u>.</li><li>The receiver can be sure that the message is generated by the receiver, because it's decryptable with its public key.</li><li>Mainly used for <b>authentication</b>.</li></ul>

## Difference between hash, MAC and DS

Hash	A (unkeyed) hash of the message, if appended to the message itself, only protects against accidental changes to the message (or the hash itself), as an attacker who modifies the message can simply calculate a new hash and use it instead of the original one. So this <b>only gives integrity</b> .
Mac Authentication Code	<p>A MAC protects against message forgery by anyone who doesn't know the secret key (shared by sender and receiver).</p> <p>This means that <b>the receiver can forge any message</b> – thus we have both integrity and authentication (as long as the receiver doesn't have a split personality), <b>but not nonrepudiation</b>.</p> <p>Also an attacker <b>could replay earlier messages</b> authenticated with the same key, so a protocol should take measures against this.</p> <ul style="list-style-type: none"><li>Inability to provide non repudiation: MACs cannot provide a proof that a message was indeed sent by the sender.</li><li>Though no third party can compute the MAC, still sender could deny having sent the message and claim that the receiver forged it, as it is impossible to determine which of the two parties computed the MAC.</li></ul>
Digital Signature	<p>A (digital) signature is created with a private key, and verified with the corresponding public key of an asymmetric key-pair.</p> <p>Only the holder of the private key can create this signature, and normally anyone knowing the public key can verify it.</p> <p>Digital signatures <b>don't prevent the replay attack</b> mentioned previously.</p> <p>So this provides all of integrity, authentication, and non-repudiation.</p>

- What they provide
  - Integrity
  - Authentication
  - Non repudiation
  - Verifiability
  - Transferability: consequence of the previous property (verifiability). A DS can be transferred b/w users.
- Scheme
  - G: key generation algorithm
  - S: signature generation algorithm
    - $S(sk, m) = \sigma$  is the signature
  - V: signature verification algorithm
- Plain RSA signature scheme

- Same key generation algorithm as PKE
- **Signing:**  $\sigma = |m^d|_n$
- **Verification:**  $m == |\sigma^e|_n$
- Why plain RSA digital signature cannot be used (RSA with padding has to be used).
  - Suffers from **existential forgery**
    - The adversary:
      - a) Knows the public key  $(n, e)$  of someone
      - b) Chooses  $\sigma$
      - c) Computes  $x = |\sigma^e|_n$   
 $x$  is random and may have no application meaning: for this reason the forgery is *existential*.
      - d) Outputs  $(x, \sigma)$
    - The receiver, in order to verify the message, performs:
      - a)  $|x^d|_n = |(\sigma^e)^d|_n = |\sigma^{ed}|_n = |\sigma|_n$ 
        - ◊  $|\sigma^{ed}|_n = |\sigma|_n$  proof is based on the Fermat's theory: this has been skipped by the lecturer.
    - In this way, the attacker is able to forge a digital signature.
      - ◆ An attacker could sign (meaningless) messages on behalf of a user.
  - Malleability
    - $\sigma_3 = \sigma_1 \cdot \sigma_2$  is a valid signature for  $m_3 = m_1 \cdot m_2$   
 Proof: decryption is  $\sigma_3^e = (\sigma_1 \cdot \sigma_2)^e = \sigma_1^e \cdot \sigma_2^e = |m_1 \cdot m_2|_n$
- ○ **Re-blocking problem** (slide 23 PDF 6.0)
  - RSA can be used for both for [public key] encryption and digital signature.
    - Only RSA has this property: **Elgamal doesn't**.
  - Let
    - Alice:
      - ◆  $(e_A, n_A)$ : public key
      - ◆  $(d_A, n_A)$ : private key
    - Bob:
      - ◆  $(e_B, n_B)$ : public key
      - ◆  $(d_B, n_B)$ : private key
  - So if Alice wants to send a crypted and authentic message, she has to send:
    - **Digital signature:**  $\sigma = |m^{d_A}|_{n_A}$
    - **Ciphertext:**  $c = |\sigma^{e_B}|_{n_B}$
  - So Alice encrypts the message with Bob's public key and signs it with her own private key.
  - What the re-blocking problem says:  $n_A < n_B$ 
    - From [https://sites.math.washington.edu/~morrow/336\\_09/papers/Yevgeny.pdf](https://sites.math.washington.edu/~morrow/336_09/papers/Yevgeny.pdf)  
 Reblocking means having to break a signed message up into smaller blocks, since the signature may be greater than the encryption (both are different, since they are from different keys from two or more distinct users). The authors of RSA, however, have provided a way to avoid reblocking a message: choose a threshold value  $h$  (say  $h = 10^{202} - 33$ ) for the public-key cryptosystem, and assure that "every user maintains two public  $(e, n)$  pairs, one for enciphering and one for signature verification, where every signature  $n$  is less than  $h$ , and every enciphering  $n$  is greater than  $h$ ". Thus, message blocking only depends on the transmitter's **signature  $n$** ."
      - $\sigma < n_A$  by definition (result of the digital signature creation  $\sigma = |m^{d_A}|_{n_A}$ )
      - $m < n_A$  is necessary to perform the digital signature encryption.
      - $\sigma < n_B$  is necessary to be able to encrypt.

◆  $c = |\sigma^{e_B}|_{n_B} = \left| \left( |m^{d_A}|_{n_A} \right)^{e_B} \right|_{n_B}$ , for this reason  $n_A < n_B$ , otherwise

the digital signature  $\sigma$  could be greater than  $n_B$ .

This is the "re-blocking problem", in other words, the authors of RSA want to avoid re-blocking the digital signature before encrypting it

◊  $|n_A| = t$

◊  $|n_B| = t + 1$

- Could be applied to messages **of any length**, by applying a sort of ECB to the message, by dividing a message into blocks and **signing all different blocks** independently.

- This is not practical because exponentiation is a computationally hard operation.
- Usually, to avoid this, **only the message digest is signed**.

#### □ Digital signature with hash functions

◆  $\sigma = |h(m)^d|_n$

- ◆ The main reason hash function are used in digital signature is for **performance**.

- ◆ Hash function properties can then influence digital signature properties.

◊ Properties recall

##### 1. Preimage resistance

*finding  $x$  given  $h(x)$*

##### 2. 2nd-preimage resistance

*given  $x$ , finding  $x' \neq x$  such that  $h(x) = h(x')$*



##### 3. Collision resistance

*finding  $(x, x')$  such that  $h(x) = h(x')$*

◊ Properties faults examples

- If the preimage resistance property is not guaranteed, **existential forgery** is possible.

- ◆ Let:  $pub\_key = (e, n)$  and  $priv\_key = (d, n)$
- ◆ The adversary:
  - ◆ Selects  $z < n$ , that is a guess digital signature of a message digest
  - ◆ Computes  $y = |z^e|_n$  (verification function, it outputs the message digest itself)
    - ◆ If  $h()$  is not preimage resistance, given  $y = h(x)$ , it's possible to find  $x$  s.t.  $y = h(x)$
  - ◆ Sends  $(x, z)$  ( $x$  is a guessed message ("existential"),  $z$  is its digsig)
    - ◆ Can claim that  $z$  is the digital signature of  $x$ .

- If the 2nd-preimage resistance property is not guaranteed, **repudiation** is possible.

- ◆ Alice and Bob want to sign  $x$  emitted by a TTP.
- ◆ Both of them sign the message with their own private key.
- ◆ If  $h()$  is not 2nd-preimage resistance, Alice may find  $x' \neq x$  s.t.  $h(x) = h(x')$ , so Alice can repudiate  $x$ .

- If the collision resistance property is not guaranteed, **fraud** can happen.

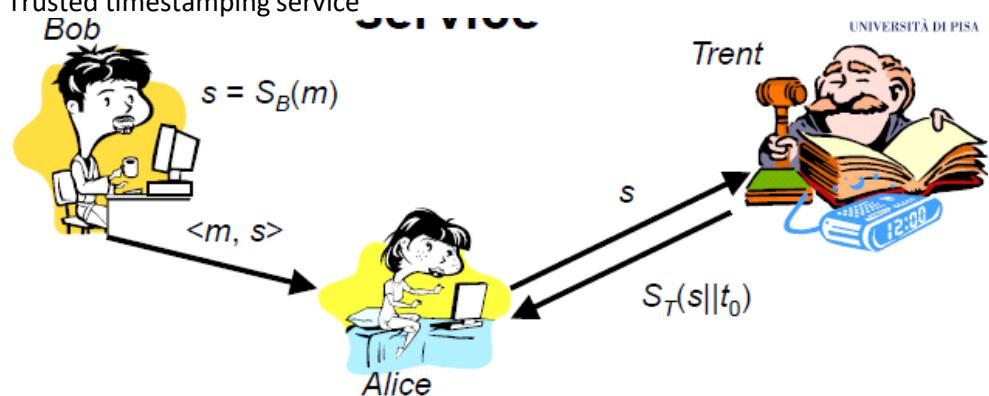
- ◆ Alice wants to sign  $x$ , received from Bob.
- ◆ If  $h()$  is not collision resistant, Bob can generate  $(x, x')$

such that  $h(x) = h(x')$ .

- ◆ Bob in this way can pretend that Alice signed  $x'$  instead of  $x$ .

- Elgamal signature scheme (DSA)
  - Two different algorithms for encryption and decryption
  - Not used anymore
  - **Elliptic Curve DSA (ECDSA)**
    - Uses smaller keys in comparison with Diffie-Hellman or RSA.
- Bad users repudiability attempts
  - Repudiation with digital signatures can be made if someone admits that its own private key has been compromised.
    - Solutions
      - Prevent direct access to the private key
 

An user receives a smartcard that contains (generates) the private key.  
The only thing that the user can do is to claim that s/he has lost the smart card.  
Losing the smart card is noticeable, meanwhile, the private key (for instance in a file) could be modified, read and sent away, and the user could not know it.
      - Trusted timestamping service



- Trent certifies that digital signature **s exists** at time  $t_0$
- If Bob's priv-key is compromised at  $t_1 > t_0$ , then **s is valid**
- ◆ This requires a **trusted** third party.
  - ◇ **Trusted Notary Service (TNS)**
    - ▶ Can also assure that **s is valid**.
    - ▶ Can certify the existence of a message (patents).
  - ◆ This is sometimes implemented by CAs.

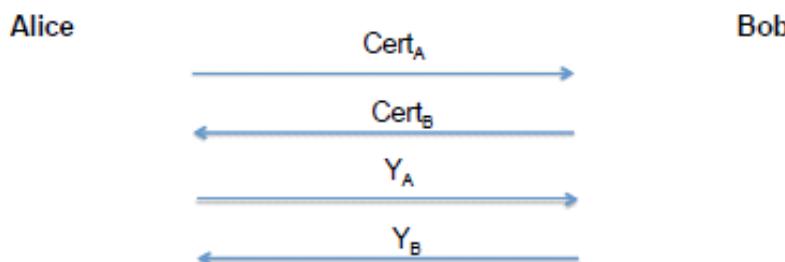
# Public key infrastructure

martedì 23 maggio 2017 11:11

- How public keys can be distributed and revoked
  - Basic key transport protocol is insecure against MITM attacks, because there's no link between users and public keys.
- **Certificate**
  - What does it contain
    - $A$ : identifier (username)
    - $e_A$ : public key
    - $L_A$ : validity period
      - Very often, this is specified by the law.
      - If key couples are used outside this time interval (**expired**), signatures have no legal value, even though they continue to have a "technical value".
      - What is it for:
        - ◆ Technical reasons: it avoids giving too much crypted material to crypto-analysts.
        - ◆ Administrative reason: encharged people have their job for a limited time.
      - When a still valid key cannot be used anymore (after firing someone, etc.) it must be **revoked**.
    - Revoking keys is difficult:
      - ◆ Who has the right of doing it?
        - ◊ The owner (for example when a user finds out that his/her own key has been compromised)
        - ◊ The CA
      - ◆ Revocation must be as quick as possible  
There shouldn't be a period of time during which the key is used: users can be offline, and so it can be hard to inform them immediately.
      - ◆ Certificate Revocation List (**CRL**)
        - ◊ The CA puts revoked keys into this list (which is public)
        - ◊ The list is signed by the CA
        - ◊ It is periodically published by the CA (with a timestamp)
    - $S_{CA}(A \parallel e_A \parallel L_A)$ : digital signature that **links together the previous three fields**.  
 $S_{CA}(H(A \parallel e_A \parallel L_A))$  it's also ok.
  - Certificate-base [Diffie-Hellman Protocol](#)

$$Y_A = g^a \pmod{p}$$
$$\text{Cert}_A = A, Y_A, L, S_{CA}(A \parallel Y_A \parallel L)$$

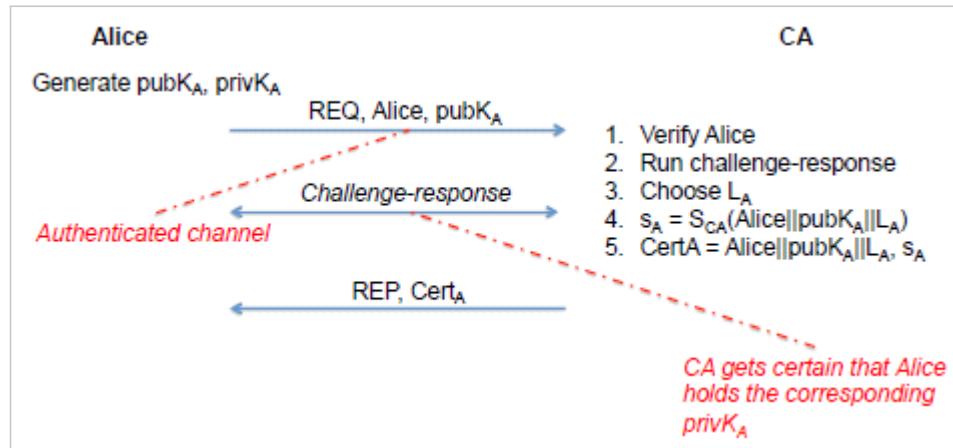
$$Y_B = g^b \pmod{p}$$
$$\text{Cert}_B = B, Y_B, L, S_{CA}(B \parallel Y_B \parallel L)$$



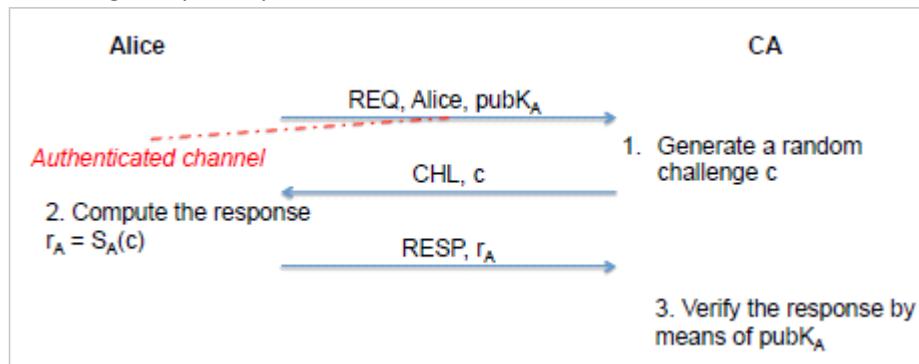
- A certificate can be sent contextually to the public key
- $Y_A$  is Alice's public key, and it's generated with DH ( $Y_A = |g^a|_p$ ).

- Generation

- With user-provided keys

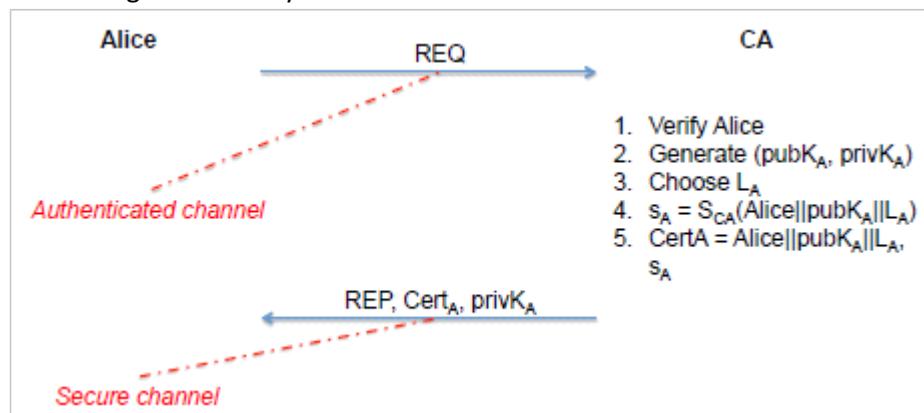


- The user sends his/her keys, that can be auto-generated.
  - ◆ Typically the public and private key couple is given by hand (e.g. bank accounts).
- This channel has to be secure because the user also has to send his/her private key, in order for the CA to check that too.
- Challenge-response protocol (CRP):



- ◆ If the challenge is not random, a **replay attack** can be performed.
- ◆ The challenge-response protocol can be implemented also with a cipher.
- ◆ No requirements about this channel, because the certificate does not contain any secret information.

- With CA-generated keys



- The second channel has to be secure because the private key has to be sent.
- Typically the private key shouldn't be given to the user (see the reasons above), so that can be given with a smart card.

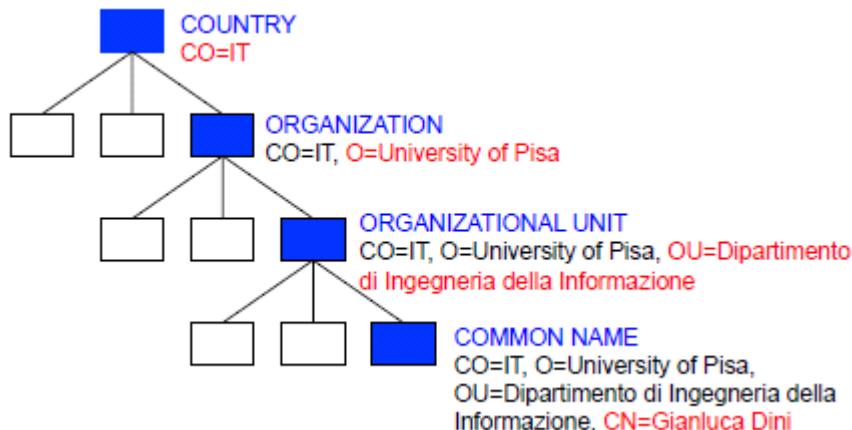
- Verification

- **Bob verifies authenticity of  $\text{pubK}_A$  using  $\text{Cert}_A$** 
  - Bob obtains CA's public key  $e_T$  [**once at set-up**]
  - Bob verifies validity of CA's public key
  - Bob verifies the digital signature in  $\text{Cert}_A$  by using CA public key
  - Bob verifies that certificate  $\text{Cert}_A$  is valid (within  $L_A$ )
  - Bob verifies that certificate  $\text{Cert}_A$  is not revoked
    - At this point, Bob has to download the proper CRL and check this.
      - ◆ For example, CRLs are published once a day: Bob, in the worst case, can discover that Alice's key has been revoked after an entire day.
      - ◆ During the 90's, browsers didn't implement this because the CRL was a few MBs long.
    - For this reason, not even CAs implemented this.
  - If all these checks are successful, then Bob accepts  $\text{pubK}_A$  as authentic key of Alice
- Key lifetime, backup and recovery: practical problems
  - **Private key backup**
    - Problem description
 

Given a couple of keys used for encryption (or to establish a secret key).  
The private key is usually in a smart card: if this one gets broken, the private key is gone, and so all the private encrypted data.  
For this reason a private key should be backed up.
    - Who can do it
      - Not the user, cannot be able to do it.
      - Not the government, that could be able to read all the encrypted data.
      - The **company** that gives the smart card has all the rights to manage the encrypted data.
    - This backup can allow a user to read encrypted data even after the certificate expires.
      - When a certificate expires, the private key should be deleted (because no one should be able to use it anymore).
      - By making a backup of a private key, a user can repudiate digitally signed messages by saying that the message was signed after the certificate expiration date.
      - All bank users (or softwares like operating system) have to install their public key into all devices (smart cards or operating systems).  
If this get lost, ...
  - **Threshold crypto**
    - A message is encrypted using a public key
    - The corresponding private key is split into  $n$  shares
    - At least  $t$  (threshold) shares are necessary to reconstruct the secret
    - The system tolerates the compromisation of  $t-n$  nodes
- CA's obligations

- CA must be reliable
  - CA must verify that the name (Alice) goes along with the key ( $\text{privK}_A$ )
  - CA must verify that owner of ( $\text{privK}$ ,  $\text{pubK}$ ) pair is really entitled to use that name
  - CA establishes rules/policies to verify that a person has rights to the name
- CA's certificate must be (immediately) available
  - Ex. CA's certificate is embedded in a browser installation package
  - Ex. CA's certificate is released at user registration time
- How to distribute CA's certificates
  - Released at registration time (banks)
  - In newspapers. They publish hashes of their certificates.
  - Embed it in a browser installation package.
    - This is insecure: a trojan can install a certificate, that just consists in a simple file in a browser's directory.
- Trust delegation
  - Bob receives a certificate about Alice's public key.  
This certificate has been signed by some CAs.
  - If Bob trusts the CA's public key, he can then trust Alice's one.
- X.509 standard
  - Data structure
    - i. Version
    - ii. Serial number
    - iii. Signature algorithm identifier
    - iv. Issuer distinguished name
    - v. Validity interval
    - vi. Subject distinguished name
    - vii. Subject public key information
 

The following two fields are needed to avoid ambiguity when the subject and the issuer have the same name:
    - viii. Issuer unique identifier (v=2,3)
    - ix. Subject unique identifier (v=2,3)
    - x. Extensions (v=3)
    - xi. Signature
  - Hierarchical names



- CN: common name
- OU: organization unit
- O: organization

- Certificate example

**Certificate name**

[www.mps.it](http://www.mps.it)

**Consorzio Operativo Gruppo MPS**

**Terms of use at [www.verisign.com/rpa](http://www.verisign.com/rpa) (c)00**

**Florence**

**Italy, IT**

**Issuer**

**VeriSign Trust Network**

**[www.verisign.com/CPS](http://www.verisign.com/CPS) Incorp. by Ref. LIABILITY LTD. (c)97 VeriSign**

- This is the CA

**Details**

**Certificate version:** 3

**Serial number:** 0x652D0F8ADAB4C7B168A27BBD1C3E9D9D

**Not valid before:** Mar 2 00:00:00 2004 GMT

**Not valid after:** Mar 2 23:59:59 2005 GMT

**Fingerprint:** (MD5) CA CA 88 08 EC D0 8E 49 A6 9A 66 C4 69 31 E0 AE

**Fingerprint:** (SHA-1) 82 64 CB 69 F0 43 86 43 FF B4 55 D4 25 EF 51 60 65 46 D3 87

- Certificates have their own version (depending on the validity period)
- The **fingerprint** is the **hash of the certificate**

**Public key algorithm:** rsaEncryption

**Public-Key (1024 bit):**

**Modulus:**

```
00: E1 80 74 5E E7 E5 54 8B DF 6D 00 95 B5 96 27 AC
10: 66 93 E0 49 B9 6F 5B 73 53 1C BE 1C EB 47 64 B2
20: 12 95 70 E6 CD 50 67 02 88 E3 EE 9D B1 91 49 C8
30: 8D 58 19 4B 86 8F C0 2E 65 E8 F2 D4 82 CC 55 DB
40: 43 BC 66 DA 44 2F 53 B3 48 4B 37 15 F3 AB 67 C1
50: 69 B4 53 23 19 30 1A 19 23 7F 28 E0 E3 C0 6B 18
60: FF 84 C4 AC A9 74 28 DB FF E9 48 CA 75 D5 35 D6
70: 46 FB 7D D4 A7 3F A1 4B 00 60 14 DC D5 00 CF C7
```

- This is n

**Exponent:**

**01 00 01**

- This is e

**Public key algorithm:** sha1WithRSAEncryption

```
00: 23 A6 FE 90 E3 D9 BB 30 69 CF 43 2C FD 4B CF 67
10: D7 3C 46 22 9A 08 DB 05 1D 45 DC 07 F3 1E 4D 1F
20: 4B 11 23 5B 42 91 14 95 25 88 1F BD 60 E5 6F 84
30: 44 70 7A 95 EC 30 E4 46 4F 37 87 F1 B2 FA 45 04
40: 6F 7C BE 97 25 C7 20 E7 F3 90 55 51 99 3A 72 35
50: 40 F2 E8 E3 36 3A 7D 58 61 9C 91 D6 AC 34 E7 E8
60: 09 27 64 4F 2C 4C C2 D2 A3 32 DB 2B 7E F0 B6 F3
70: 69 96 E4 2B C3 2B 42 ED CA 2C 3C C8 F5 AA E6 71
```

**Extensions:**

**X509v3 Basic Constraints:** CA:TRUE, pathlen:0

**X509v3 Certificate Policies:**

**Policy:** 2.16.840.1.113733.1.7.1.1

**CPS:** [https://www.verisign.com/CPS](http://www.verisign.com/CPS)

**X509v3 Extended Key Usage:** TLS Web Server Authentication, TLS Web Client

Authentication, Netscape Server Gated Crypto, 2.16.840.1.113733.1.8.1

**X509v3 Key Usage:** Certificate Sign, CRL Sign

**Netscape Cert Type:** SSL CA, S/MIME CA

**X509v3 CRL Distribution Points:**

**URI:** <http://crl.verisign.com/pca3.crl>

- **Basic constraint CA:** identifies if the subject of certificates is a CA who is allowed to issue child certificates.

- The private key used to sign this certificate cannot be used to sign other certificates
  - If I receive a certificate signed by CA's private key, with the FALSE value I can assure that the digital signature is not valid because the server mps.it has not respected all the constraints.

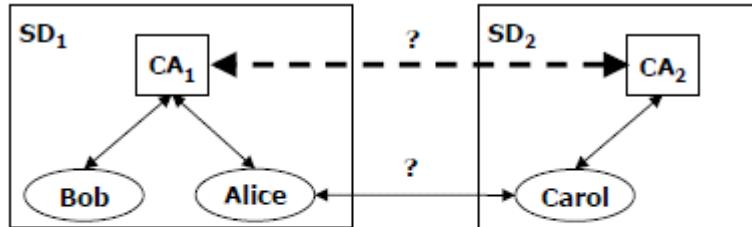
- **CRL Distribution Points:** gives the address of the CRL.

It cannot be modified because it is protected with the digital signature in the certificate.



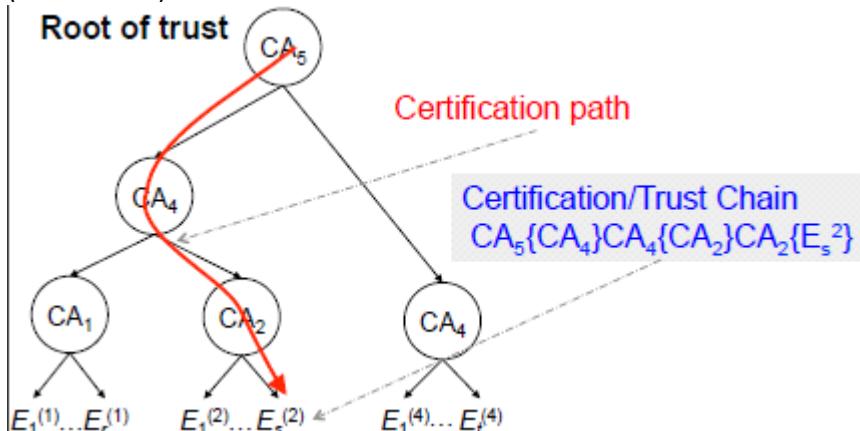
- In CRLs there are no entire certificates in it for privacy reasons: if certificates get published, also names are published (and so, information about company employees and companies ).  
It just contains a serial number.

- Multiple CAs
  - Problem



In order to let Alice and Carol communicate, it is necessary to establish **a trust relationship** between the Certification Authorities

- (Hierarchical) Centralized trust model



- A user, in order to check another user's certificate, has to trust the root CA.  
Once a user trusts the root CA, it can then verify all the other CAs contained in the chain.

- So **every has to trust the root CA**.
  - Also all the rules are stated by the root CA.
- This is used by MasterCard: all their smartcards contain the root CA's certificate.
  - This is why their cards can be used all around the world.
- Constraints about this structure
  - Constraint on the **chain length**
  - Constraint on the **set of domains** (width of CA's tree)

Those constraints are specified in the standard.

- The root CA certifies itself.

X.509 example:

Certificate name  
VeriSign Trust Network  
[www.verisign.com/CPS\\_Incorp/by\\_Ref LIABILITY LTD.\(c\)97 VeriSign](http://www.verisign.com/CPS_Incorp/by_Ref LIABILITY LTD.(c)97 VeriSign)

Issuer  
VeriSign, Inc.  
Class 3 Public Primary Certification Authority  
US

UNIVI

**Extensions:**

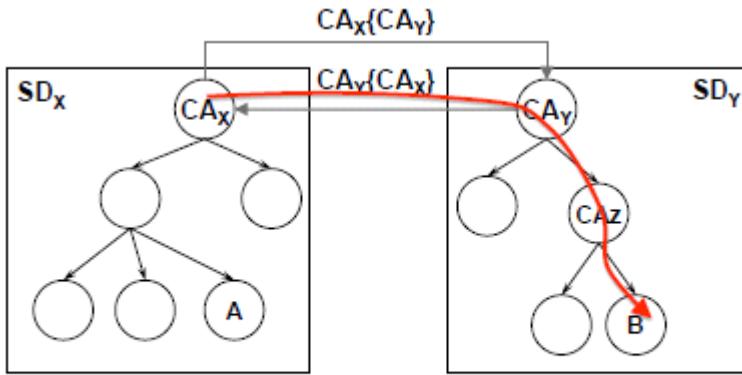
- X509v3 Basic Constraints: CA:TRUE, pathlen:0
- X509v3 Certificate Policies:
- Policy: 2.16.840.1.113733.1.7.1.1
- CPS: <https://www.verisign.com/CPS>
- X509v3 Extended Key Usage: TLS Web Server Authentication, TLS Web Client Authentication, Netscape Server Gated Crypto, 2.16.840.1.113733.1.8.1
- X509v3 Key Usage: Certificate Sign, CRL Sign
- Netscape Cert Type: SSL CA, S/MIME CA
- X509v3 CRL Distribution Points:
- URI:<http://crl.verisign.com/pca3.crl>

- This time, Basic Constraints CA is set to TRUE.
- Also note that pathlen is zero.
- Certification Practice Statement (CPS): document, defined by the root CA, that describes how all the certification method is implemented.

The CA has to implement the CPS.

- Cross-certification

- Companies do not want to trust a root CA, and they don't want their CA to be placed in the second-level.
- Structure



**Chain of Certificates**  
CA<sub>x</sub>{CA<sub>y</sub>}CA<sub>y</sub>{CA<sub>z</sub>}CA<sub>z</sub>{B}

- X and Y have to certify themselves.

- Enterprise model

- More than one company can co-operate between themselves.  
Every root CA has to make a deal with any other root CA.
- No hierarchy

- Hub-and-spoke model

- A few company root CAs choose another root CA.

- Browser model

- CAs are already installed in the browser bundle.
  - In this way, all users that download it can trust those CAs.
  - The number of existing CAs is around 650, but just around 75 are installed in browser bundles.

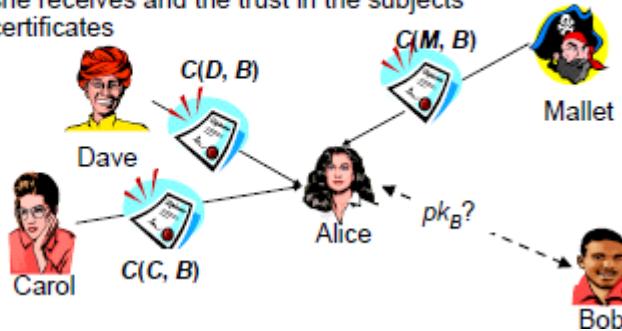
- CA incidents

- All those cases are MITM attacks.

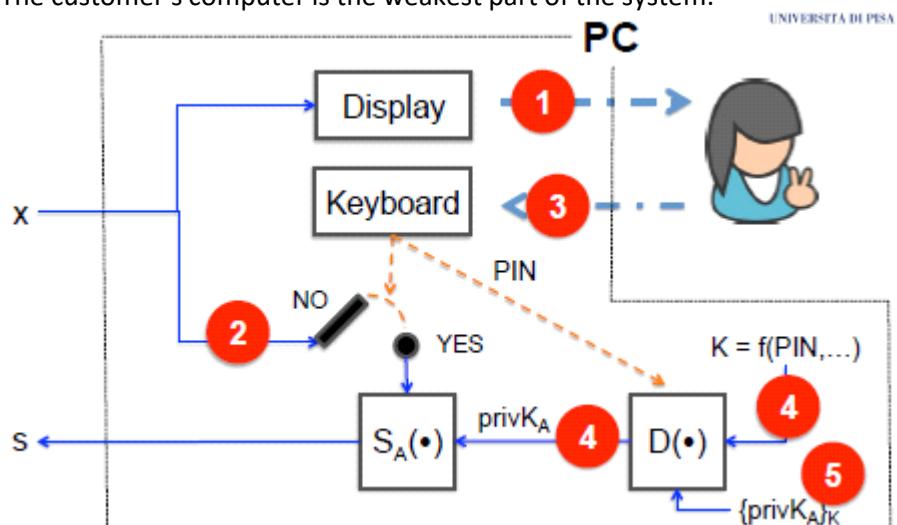
- Compromization
  - Mainly because CAs try to minimize security features in order to earn more (*DigiNotar*)
  - When CA companies trust some given certificate, the adversary who gave the certificate to the CA can intercept an HTTPS or SSL connection and act like the desired server (*Iranian attack*) (servers can do this with proxies that generates certificates on-the-fly, signed with a trusted CA already installed in the browser)

- boundle).
  - ◆ This can be used to decrypt secret messages and to poison DNS servers.
- Misconfiguration (*Turktrust*)
  - This happens when CAs sells certificates to another intermediate CA **and give them the possibility to sign as they were the initial CA**, that can perform MITM attacks.
- Solutions
  - **Public key pinning**
    - Chrome carries with itself a list of known-good certificates (and a list of presumed-good CAs), that are used as a "standard model" for each common server.
    - When the user surfs the web and gets a certificate, thanks to the public key pinning, Chrome can check if the received certificate matches all the rules contained in the pre-installed list of certificates.
  - Certificate transparency
    - Normally, when CAs give a certificate, they don't reveal this fact (for business reasons).
    - The ct countermeasurement proposes to release certificates in a public database. CAs do not want to do this.
  - Convergence
    - Download a certification directly and from a set of trusted CAs and **compare them** (for instance ).
  - DANE (DNS-based Authentication of Name Entities)
    - It consists in storing public keys in a DNS record.
  - Extended validation certificates
    - ? □ Prove the legal entity controlling the website or software package...
    - ? □ ...promise what we were promised a decade ago and we never got.
  - Revocation options
    - CRL (Certificate Revocation List)
      - ◆ Offline
    - OCSP (Online Certificate Status Protocol)
      - ◆ Online
      - ◆ Pros
        - ◊ Lighter in respect to CRL
      - ◆ Cons
        - ◊ The protocol is totally in the clear, so everyone sniffing can know what certificates are requested.
        - ? ◊ Exposed to replay attack.
        - ◊ Most of browsers ignore OCSP timeouts and establish the connection as well: a MITM attack can be then performed.  
So OCSP can be **useful just for update/patch releases**.
  - Configure browsers in order to exploit certificates
- What browsers do in reality
  - Not always they connect to the CRL/OCSP to check if the certificate is revoked or not. This is because this "blocks the browser experience".
  - This is encouraged when the revocation infrastructure is unreachable.
  - Different types of server certificates are then created (this changes the colour of the certificate next to the browser URL):
    - DV and OV: browsers do not check if those kind of certificates have been revoked.
    - Extended Verification (DV): the browser tries to connect to the revocation infrastructure, but for whatever reason it's not available, they just ignore that.
- Pretty Good Privacy (PGP)

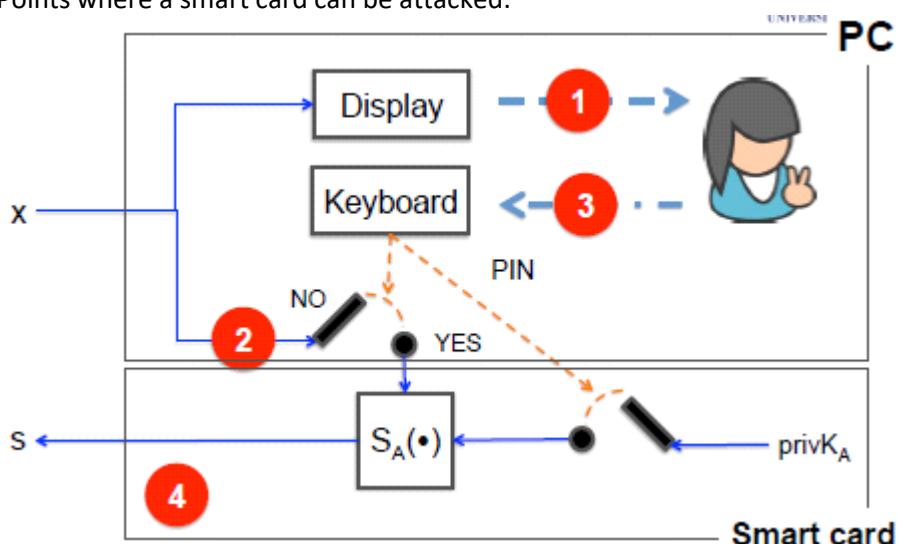
- It provides **confidentiality** and **integrity** for emails.
- It uses the Trust model
  - The user decides how much trust to put in a certificate
    - Alice determines the trust in  $pk_B$  according to the number of certificates she receives and the trust in the subjects issuing the certificates



- All the decisions are made by Alice, and not by a root CA.
- Alice can have different shades of trustness:
  - Complete trust
  - Marginal trust
  - No trust
- A key is valid if it has been signed by:
  - A completely trusted key
  - By two marginally trusted keys
- System considerations on digital signatures for e-commerce
  - **Non-repudiation** is fundamental.
    - Reasoning (pretty obvious):
      1. The owner keeps secret the  $privK$
      2. If  $\forall (pubK, s) == \text{True}$  then  $s$  was made by  $privK$
      3. A valid, unrevoked certificate links  $pubK$  to a name
      4. Certification process links name to owner
      5. Therefore the owner has generated signature  $s$   
(and he is responsible of it)
    - Points that can be attacked (in red):
      - The customer's computer is the weakest part of the system:



1. The attacker shows Alice  $x'$  instead of  $x$
  2. The attacker shows Alice  $x$  but signs  $x'$
  3. The attacker changes Alice's decision or steals Alice's PIN
  4. The attacker steals the private key
  5. The attacker steals the encrypted private key for an offline attack
1. The attacker can display something else to Alice
  2. Another command can be given in the back side
  3. Another command can be given in the front end
  4. The malicious software can intercept the PIN and can use it to make stuff.
  5. The malicious software can try to discover the key  $K$  (and it's easy because it depends on the PIN)
- Points where a smart card can be attacked:



- The adversary could still steal the user's smart card.
  1. The attacker shows Alice  $x'$  instead of  $x$  and trick Alice into signing  $x$
  2. The attacker replaces  $x$  with  $x'$
  3. The attacker changes Alice's decision or steals Alice's PIN
  4. The attacker steals and attacks the smart card
    1. Physical attack
    2. Side-channel attack
- A smart card can be integrated in a CPU (like in the ARM processors, they have secure dedicated processors). This can be done in order to prevent some malicious intrusion between the normal CPU and the secure one, by making the communication between those two parts in hardware. The software run by the secure CPU is considered more secure because this one is unaccessible, nothing can be installed on it and everything is tested.

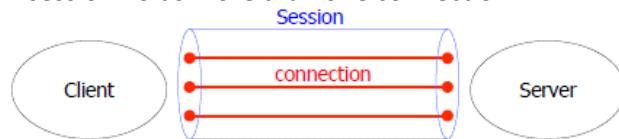
# SSL

martedì 23 maggio 2017 11:10

- ★ • Authentication
- ★ • Secrecy
- ★ • Integrity

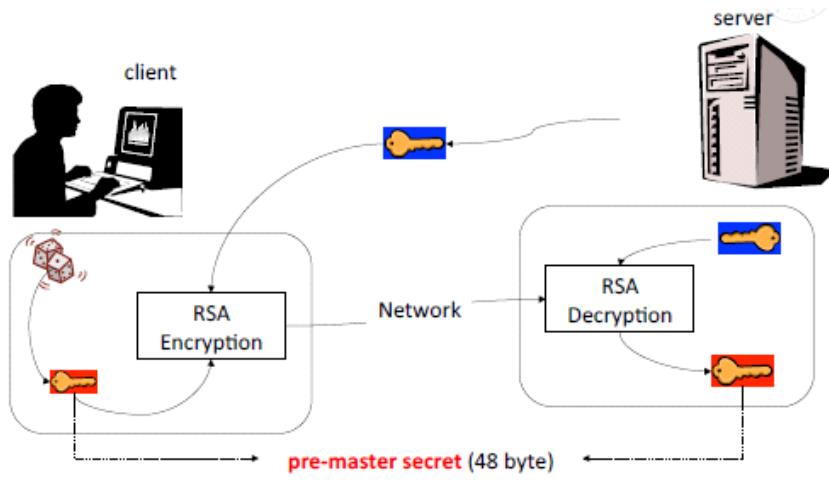
- ! • Transport layer (above TCP)
  - SSL vs TLS
    - They are almost equal
    - The only thing that changes is the implementation (hashing functions, etc.)
    - They cannot inter-operate b/w themselves

- Session
  - A session holds more than one connection

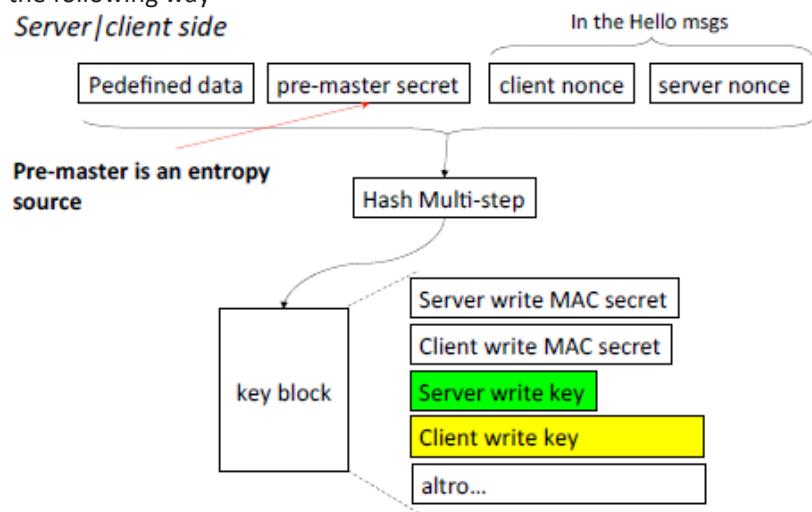


- State
  - Session ID
  - Peer certificate
  - Compression method
  - Cipher spec
  - Pre-master secret (48 Bytes)
- After the pre-master secret establishment (computationally heavy, made with PKE), the session is formed.
- This pre-master secret (**PMS**) lasts all session long (like other secrecy options). In this way, this expensive negotiation is made just once.
  - A session has a certain lifetime.
- Both sides create several keys from the PMS used in all subsequent connections.
- ? ◦ One key for each direction

- ! • Composed on 4 protocols
  - **Change cipher** protocol
  - **Alert** protocol
    - Whenever the client or the server notice something strange.
    - Can be "fatal", that will end up the session.
  - **Handshake** protocol
    - Establishes a secure session
      - Mutual **authentication**
      - Cipher suite negotiation
        - ? ◆ Key establishment
          - ◆ Encryption scheme
          - ◆ MAC
      - ? □ Both parties establish a **pre-master secret** (PMS)
      - Basic scheme (from [Wikipedia](#))

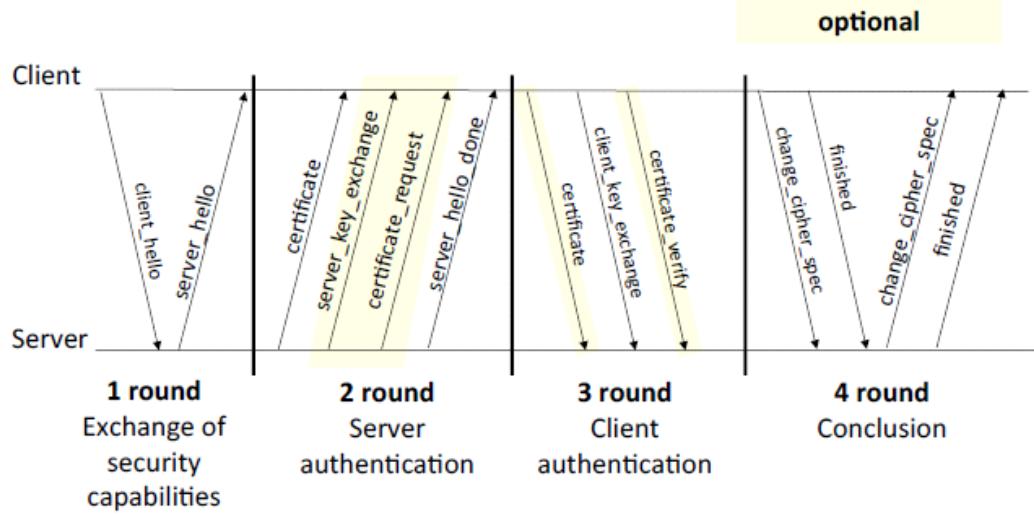


- **client\_hello:** The client requests a secure connection and presents a list of supported [cipher suites \(ciphers and hash functions\)](#).
- **server\_hello:** From this list, the server picks a cipher and hash function that it also supports and notifies the client of the decision.
- **certificate:** The server also sends its [certificate](#).
- The client confirms the validity of the certificate before proceeding.
- To generate the [session keys](#) used for the secure connection, the client either:
  - ◆ encrypts a **48B random number** (the pre-master secret **PMS**) with the server's public key and sends the result to the server (which only the server should be able to decrypt with its private key); both parties then use the random number (PMS) to generate a unique **session key** for subsequent encryption and decryption of data during the session
    - ◊ The PMS belongs to the session
    - ◊ A new session key is computed (from the PMS) for each connection in the following way



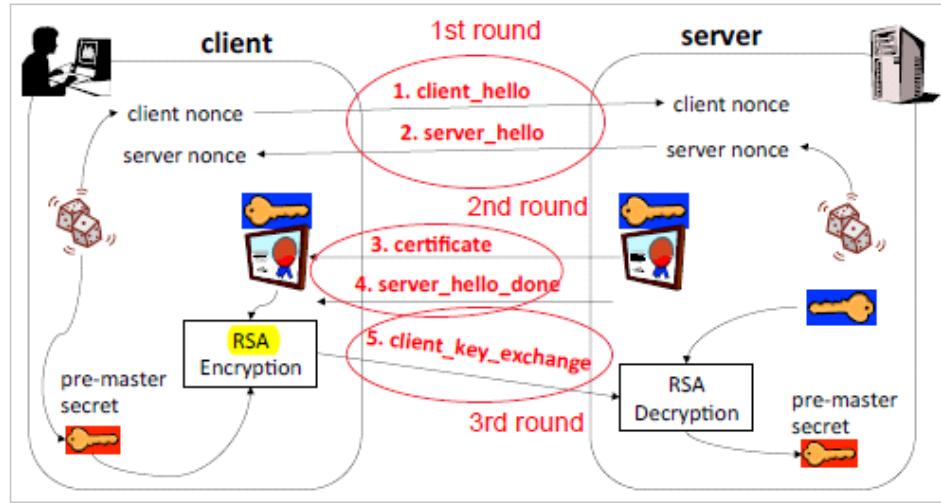
- ◆ uses [Diffie-Hellman key exchange](#) to securely generate a random and unique session key for encryption and decryption that has the additional property of forward secrecy: if the server's private key is disclosed in future, it cannot be used to decrypt the current session, even if the session is intercepted and recorded by a third party.
- **Authentication**
  - Optional client authentication
    - ◆ The server can send a `certificate_request` to the client.
    - ◆ The client is then required to send:
      - ◊ `certificate`
      - ◊ `certificate_verify`

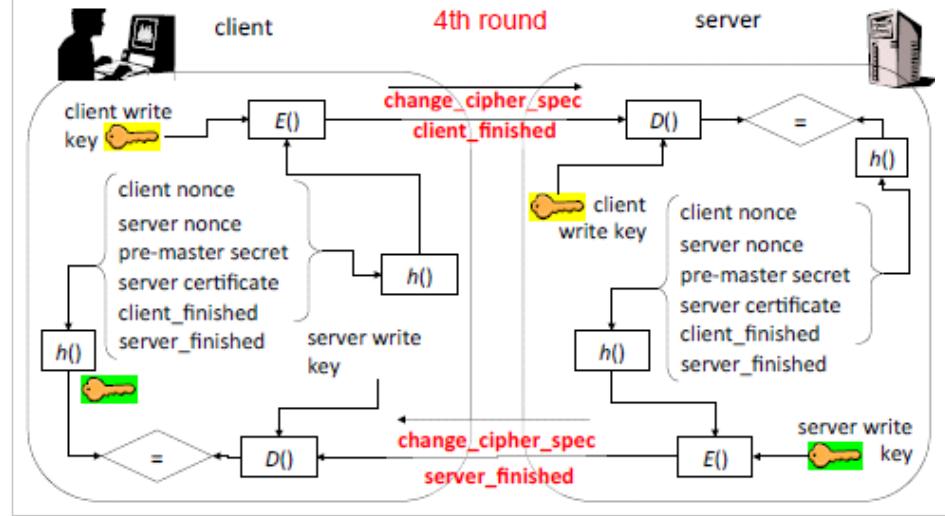
- ▶ Certification **confirmation**: hash of some known data (certificates, nonces and the PMS), encrypted with his/her private key.
- Mandatory server authentication
  - ◆ The client can trust the server because it can have its certificate.
  - ◆ The channel is secure thanks to SSL
    - ◊ Confidentiality: cipher
    - ◊ Authentication: MAC
  - ◆ The client can then send the password in the channel.
- All messages



- **Mandatory messages in detail**

- Overall scheme





1. Round 1

a. **C → S client\_hello**

- ◊ SSL version
- ◊ 32 bits timestamp
- ◊ Random 28B quantity (nonce)
- ◊ Session ID
  - ▶ 0: the server understands that the client wants to initiate a new session
  - ▶ Otherwise it wants to create a new connection within the session
- ◊ Cipher suite
  - ▶ List of algorithm triples:
    - ◆ Key establishment (RSA, DH, Ephemeral DH, ...)
    - ◆ Cipher (RC4, DES, AES, ...)
    - ◆ cipher type
    - ◆ IV size
    - ◆ isExportable (b/w countries)
    - ◆ MAC (MD5, SHA-1, ...)
    - ◆ MAC size
    - ◆ [ Key material, other infos for the key generation ]
  - ▶ Some tuples are standard  
Like SSL\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA
- ◊ Compression method

b. **S → C server\_hello**

- ◊ The server agrees on the algorithms

2. Round 2

a. **S → C certificate**

- ◊ If it's not sent, a fatal alarm will be sent.
- ◊ The client checks the validity of the server's certificate before proceeding.

b. **S → C server\_hello\_done**

3. Round 3

a. **C → S client\_key\_exchange**

- ◊ Key exchange data  
Depends on the chosen key exchange protocol with the hello

messages:

- ▶ RSA
  - ▶ DH
    - ◆  $p, g, Y_c, Y_s$
  - ▶ Anonymous or ephemeral DH
  - ▶ Fixed DH
- ◊ The client encrypts a 48B random PMS by means of the server's public key: after this message, both parties have the PMS.
- ▶ Both sides can now generate their own keys for every connection in this way.

#### 4. Round 4

a. 

C	→ S	change_cipher_spec
---	-----	--------------------

- ◊ The client proves the server that he actually has all the correct generated keys.  
It can prove it by sending encrypted already-known stuff.

b. 

C	→ S	client_finished
---	-----	-----------------

c. 

S	→ C	change_cipher_spec
---	-----	--------------------

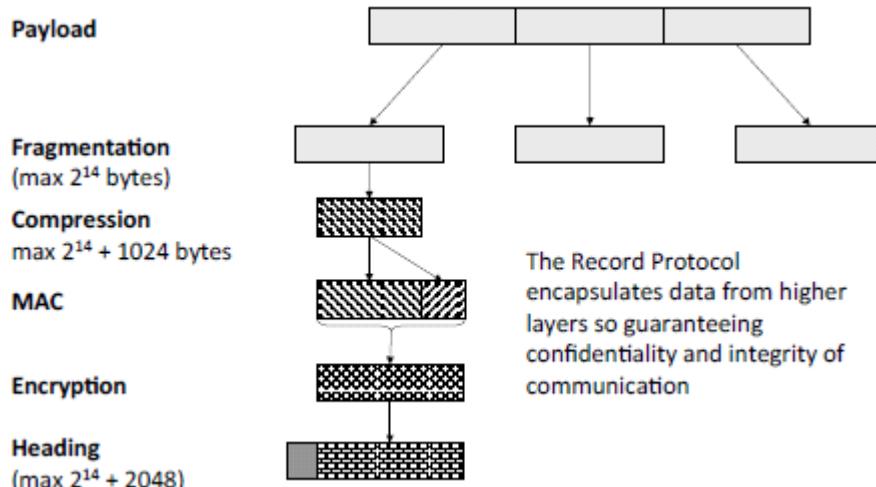
- ◊ Same thing of the change\_cipher\_spec message.

d. 

S	→ C	server_finished
---	-----	-----------------

#### o Record protocol

- Fragments, compresses, MACs and encrypts from high protocol stack's levels.
- Scheme



- o The payload is divided in  $2^{14}$ B fragments
- o The fragment is then compressed (default behaviour: no compression)
  - The compression has to be lossless
- o It computes the MAC
  - Using the algorithm established during the handshake protocol
- o The compressed fragment and the MAC are then encrypted
  - Using the algorithm established during the handshake protocol
- o Then an header is added
  - Payload types:
    - Application payload
    - Alert payload
      - ◆ Fatal exceptions (the session is ended)
      - ◆ Other non-fatal exceptions (managed by someone else)
      - ◊ Unknown certificates

TLS defines more alarms.

- SSL security
  - Nonces in hello messages
    - Fresh master secret
    - Avoid replay attacks
  - Certificates
    - Avoid MIM
  - PMS and nonces must be unpredictable
- Vulnerabilities were found in some implementations
  - PMS PRNG in SSL 2
    - It used the hash of:
      - Time of the day (guessable)
      - Process ID (the range was restricted)
      - Parent process ID (typically 0 or 1)
    - All of those three quantities could have been guessed by an attacker.  
Could be broken in 25 seconds.
  - Heartbleed attack
    - Exploited a buffer-overflow vulnerability.
- Phishing
  - In this case the malicious website sends a **valid** certificate.
  - Normal users do not understand certificates.
- E-payments
  - The credit card number is a public information
  - For this reason also the CCV2 is needed during a payment check.
  - When it is not needed, the Italian law determines how's going to be penalized (the merchant) because there's no technical way to determine how has paid.
  - **Secure Electronic Transactions (SET)**

# Analysis and design of cryptographic protocols

martedì 23 maggio 2017 11:10



- 08.ban-logi
- C
- Session key (K) establishment
  - Summary
    - A session key K is used for one communication session, to encrypt a large amount of data just once.
    - A long term key W is used for many runs of the key establishment protocol.  
In each round, it encrypts a small amount of data.
  - Three flavors
    - One-pass
      - $A \rightarrow B: E(W, t_A \parallel "B, A" \parallel K)$
      - ◆  $t_A$  is a timestamp (necessary to avoid the replay attack).
        - ◊ It then requires (at least loosely) synchronized clocks.
        - ◊ Instead of it (because it is not easy to achieve), a nonce can be used.
    - With challenge-response
      - $B \rightarrow A: n_B$
      - $A \rightarrow B: E(W, n_B \parallel "B, A" \parallel K)$
      - ◆  $n_B$  is a fresh quantity not used before, randomly generated by B.  
It could be a counter or else.
    - Both parties contribute to the session key
      - One party cannot always trust the other party's random generator for creating the session key K.
      - The two established session keys are then **XORed to each other**: its property assures that if at least one entity is random, then the result is also random (see paper notes).

# The BAN logic



- Practical tool that allows to verify security protocols, by verifying design principle that should be followed during a design of a security protocol.
    - It can only prove that a protocol is correct.
    - It can only prove ideal correctness: problems can still occur in implementation.
  - Formalism

- ### ○ Examples

$A \stackrel{\kappa}{\equiv} T \Rightarrow A \leftrightarrow B$  A believes T an authority on generating session keys

$A \mid\equiv T \Rightarrow \# \left( \begin{smallmatrix} K \\ A \leftrightarrow B \end{smallmatrix} \right)$  A believes that T is competent in generating fresh session keys

- Preliminaries
    - Past and present (from the start of the protocol) epochs
      - Beliefs achieved in the present are stable for all the protocol duration.
      - Beliefs of the past may not hold in the present.
    - A says  $X \Rightarrow A| \equiv X$ 
      - Parties are believed to behave correctly.
  - Postulates (or this with the formalism introduced before)

Message meaning (same result)
For shared keys:
$B  \equiv A \xleftrightarrow{K_{ab}} B, B \lhd \{X\}_{K_{ab}}$
$\underline{B  \equiv A  \sim X}$
For public keys:
$B  \equiv \overset{K_a^+}{\underset{\mapsto}{\sim}} A, B \lhd \{X\}_{K_a^{-1}}$
$\underline{B  \equiv A  \sim X}$
For shared secrets:
$B  \equiv A \xrightleftharpoons{K} B, B \lhd \langle X \rangle_K$
$\underline{B  \equiv A  \sim X}$

- ## 2 Nonce verification

$B \models \#(X), B \models A \mid \sim X$	The result is: B believes A has sent X in this protocol execution instance.
$B \models A \models X$	The final result is given thanks to <a href="#">this property</a> ( $A \text{ says } X \implies A \models X$ ).

- ### 3 Jurisdiction rule

$$\frac{B \models A \models X, B \models A \Rightarrow X}{B \models X} \text{ Because P believes Q is an authority on X, so "P trusts Q"}$$

- Other postulates

$\frac{P \models \#(X)}{P \models \#(X, Y)}$	$\frac{P \triangleleft \langle X \rangle_Y}{P \triangleleft X}$	
$P \models^K Q \leftrightarrow P, \quad P \triangleleft \{X\}_K$	$P \models^K \rightarrow P, \quad P \triangleleft \{X\}_K$	$P \models^K \rightarrow Q, \quad P \triangleleft \{X\}_{K^{-1}}$

- Idealizing a protocol

Real protocol	Idealized protocol	Derived assertion
$A \rightarrow B : \{A, K_{ab}\}_{K_{bs}}$	$A \rightarrow B : \left\{ A \xleftrightarrow{K_{ab}} B \right\}_{K_{bs}}$	$B \triangleleft A \xleftrightarrow{K_{ab}} B$
$B \rightarrow A : \{N_b\}_{K_{ab}}$	$B \rightarrow A : A \xleftrightarrow{K_{ab}} B$	Key confirmation for A

- When idealizing a protocol:

**Principle 1.** We have to specify the meaning of each message; specification must depend on the message contents; it must be possible to write a sentence describing such a meaning

- Protocol analysis

- Steps
  - Idealize protocol
  - Assumptions

**Principle 2.** Designer must know the trust relationships upon which the protocol is based. He/she must know why they are necessary. Such reasons must be made explicit.

- Postulates to each protocol step and determine beliefs achieved by principals at each step
- Draw conclusions

- Example

$A \models A \xleftrightarrow{K} B$	+	$A \rightarrow B : \{X\}_K$	$\Rightarrow$ <i>first message meaning postulate</i>	$B \models A \sim X$
		This is: $B \triangleleft \{X\}_K$		

- Typical analysis objectives

- Key authentication

There's no guarantee that Bob has actually the key K.

- $A \models A \leftarrow K \rightarrow B$
- $B \models A \leftarrow K \rightarrow B$

- Key confirmation

- $A \models B \models A \leftarrow K \rightarrow B$

Alice has a proof that Bob believes that K is the shared key.

This is **generally obtained by encrypting a known quantity**.

- $B \models A \models A \leftarrow K \rightarrow B$

- Key freshness

- $A \models \#(A \leftarrow K \rightarrow B)$
- $B \models \#(A \leftarrow K \rightarrow B)$

- Interaction with a certification authority

- $A \models \xrightarrow{e_B} P$

A wants to believe that  $e_B$  is Bob's public key.

- The real [Needham-Schroeder protocol](#)

- This protocol involves a TTP (T by the professor, S by Wikipedia)
- Each user share a long term secret b/w himself and the TTP ( $K_A$  and  $K_B$ )
- Objective: establish a session key  $K_{AB}$  between two users starting from long-term secrets with the TTP
- Idealization

**Real protocol**

M1	$A \rightarrow T$	$A, B, N_a$
M2	$T \rightarrow A$	$E_{K_a}(N_a, B, K_{ab}, E_{K_b}(K_{ab}, A))$
M3	$A \rightarrow B$	$E_{K_b}(K_{ab}, A)$
M4	$B \rightarrow A$	$E_{K_{ab}}(N_b)$
M5	$A \rightarrow B$	$E_{K_{ab}}(N_b - 1)$

$N_b - 1$  in M5 distinguishes challenge to response, according to Principle 10:

**Principle 10.** The contents of a message must allow us to determine:

- (i) the protocol the message belongs to, (ii) the execution instance of the protocol, (iii) the number of the message within the protocol

### Idealized protocol

M1	
M2	$T \rightarrow A: \{N_a, (A \xleftarrow{K_{ab}} B), \#(A \xleftarrow{K_{ab}} B), \{A \xleftarrow{K_{ab}} B\}_{K_b}\}_{K_a}$
	The key freshness part is not explicitly derived from the real protocol. It's implicit because T has replied using the same $N_a$ used by A in M1: that's why A believes that M2 (and therefore the key) is fresh.
M3	$A \rightarrow B: \{A \xleftarrow{K_{ab}} B\}_{K_b}$
M4	$B \rightarrow A: A \xleftarrow{K_{ab}} B$ key confirmation for A
M5	$A \rightarrow B: \{N_b, A \xleftarrow{K_{ab}} B\}_{K_{ab}}$ key confirmation for B

- o Secrets assumptions

$$A \models A \xleftrightarrow{K_a} T \quad B \models B \xleftrightarrow{K_b} T$$

$$T \models A \xleftrightarrow{K_a} T \quad T \models B \xleftrightarrow{K_b} T$$

$$T \models A \xleftrightarrow{K_{ab}} B$$

$$T \models \#(A \xleftrightarrow{K_{ab}} B)$$

- o Freshness assumptions

- $A \models \#(N_a)$   $B \models \#(N_b)$

- $B \models \#(A \xleftarrow{K_{ab}} B)$

- See why

- This is the only non-reasonable assumption.

- Let's suppose that an adversary is able to find (compromise) one session key  $K_{ab}$ .

- The adversary is also able to record the related M3 message.

- ◆ This means that the adversary is able to impersonate Alice whenever s/he likes.

- ◆ It is sufficient that one session key is compromised to break **the entire system (and not only the session)**.

- Bob has to believe that  $K_{ab}$  is fresh, but he cannot prove it.

- o Trust assumptions

Both believe that the TTP can generate shared key so they can use them.

- $A \models T \Rightarrow (A \xleftarrow{-} K_{ab} \rightarrow B)$

- $B \models T \Rightarrow (A \xleftarrow{-} K_{ab} \rightarrow B)$

- $A \models T \Rightarrow \#(A \xleftarrow{-} K_{ab} \rightarrow B)$

- o Analysis

- Goals to achieve

- Key authentication

- ◆  $A \models A \xleftarrow{-} K_{ab} \rightarrow B$

Alice believes that  $K_{ab}$  is the shared key.

- ◆  $B \mid = A <- K_{ab} \rightarrow B$   
Bob believes that  $K_{ab}$  is the shared key.
    - Key confirmation
      - ◆  $A \mid = B \mid = A <- K_{ab} \rightarrow B$
      - ◆  $B \mid = A \mid = A <- K_{ab} \rightarrow B$

Since the design of the protocol put M4 and M5, every party want to prove the other that s/he has the key.
  - By analyzing message M2
    - From the first postulate (1a)
      - ◆  $A \mid \equiv T \mid \sim \{N_a, (A \xleftrightarrow{K_{ab}} B), \#(A \xleftrightarrow{K_{ab}} B)\}$   
Because M2 (directed to A) was encrypted by means of  $K_a$ , shared key between A and T.
    - From the second postulate
      - ◆ If something fresh is received, the user believes that the quantity belongs to the current execution of the protocol.
      - $$\frac{B \mid \equiv \#(X), B \mid \equiv A \mid \sim X}{B \mid \equiv A \mid \equiv X}$$
      - ◆ Alice can then be sure that that message is not a replay, because M2 contains  $N_a$  generated by A in M1.
      - ◆  $A \mid \equiv T \mid \equiv \{N_a, (A \xleftrightarrow{K_{ab}} B), \#(A \xleftrightarrow{K_{ab}} B)\}$ 
        - ◊ So  $\mid \sim$  becomes  $\mid \equiv$  only thanks to the second postulate (nonce verification).
        - ◊ So Alice believes each thing separately
          - $A \mid \equiv T \mid \equiv A \xleftrightarrow{K_{ab}} B$
          - $A \mid \equiv T \mid \equiv \#(A \xleftrightarrow{K_{ab}} B)$

**Principle 3.** A key may have been used recently to encrypt a nonce but it may be old or compromised. The recent use of a key does not make it more secure
  - ◆ Because Alice considers T an authority on establishing symmetric keys (jurisdiction rule, third postulate)
  - ◊  $\frac{\square}{\checkmark}$  ► First goal (key authentication) for A achieved.
- By analyzing message M3
  - Bob believes that  $A \xleftrightarrow{K_{ab}} B$  comes from T (even if the message comes from Alice, that acts as a forwarding node)  
Thanks to the first postulate (1a):
 
$$B \mid \equiv T \mid \sim A \xleftrightarrow{K_{ab}} B$$

Because M3 it's encrypted by means of  $K_b$ .
  - B should have a proof that  $A \xleftrightarrow{K_{ab}} B$  is fresh, and the message does not contain any fresh quantity.
    - ◆ Another assumption about freshness has then to be added:  $B \mid \equiv \#(A \xleftrightarrow{K_{ab}} B)$  Bob has to believe that the session key is fresh.
  - After that this assumption has been added,  $B \mid \equiv T \mid \sim A \xleftrightarrow{K_{ab}} B$  becomes  $B \mid \equiv T \mid \equiv A \xleftrightarrow{K_{ab}} B$
  - We can now say that  $B \mid \equiv A \xleftrightarrow{K_{ab}} B$  thanks to the jurisdiction rule (third postulate).
    - ◊  $\checkmark$  ◆ First goal (key authentication) for B achieved.
- By analyzing message M4
  - By applying the first postulate
 
$$\frac{A \mid \equiv A \xleftrightarrow{K_{ab}} B, A \lhd \{A \xleftrightarrow{K_{ab}} B\}_{K_{ab}}}{A \mid \equiv B \mid \sim A \xleftrightarrow{K_{ab}} B}$$
  - Once again, apparently Alice doesn't have any proof about the freshness of  $A \xleftrightarrow{K_{ab}} B$ , but thanks to  $\#(A \xleftrightarrow{K_{ab}} B)$  (in M2) (third postulate: nonce verification), we can now say that:
 
$$\frac{A \mid \equiv \#(A \xleftrightarrow{K_{ab}} B), A \mid \equiv B \mid \sim A \xleftrightarrow{K_{ab}} B}{A \mid \equiv B \mid \equiv A \xleftrightarrow{K_{ab}} B}$$
    - ◊  $\checkmark$  ◆ Second goal (key confirmation) for A achieved.
    - $T \mid \equiv \#(A \xleftrightarrow{K_{ab}} B)$  becomes an assumption, because if T says something, s/he believes it
      - ◆ (thanks to [this property](#):  $A$  says  $X \implies A \mid \equiv X$ ).
- By analyzing message M5

- Like in M4 (except that for this time is for B):  $B \equiv A \sim (N_b, A \xrightarrow{K_{ab}} B)$ 
  - ◆ Thanks to the second postulate, Bob believes that the message is fresh, so also in this case "once said" becomes "s/he believes" (like before)
- $B \equiv A \equiv (N_b, A \xrightarrow{K_{ab}} B)$
- ✓ ◊ By splitting the message, the second goal (key confirmation) for B is achieved.

- Otway-Rees protocol



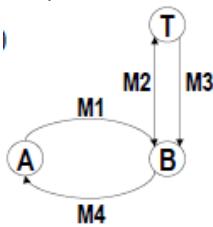
- Otway-Rees  
s BAN logic
- Real protocol
  - M1.  $A \rightarrow B: M, A, B, E_{K_A}(N_A, M, A, B)$
  - M2.  $B \rightarrow T: M, A, B, E_{K_A}(N_A, M, A, B), E_{K_B}(N_B, M, A, B)$
  - M3.  $T \rightarrow B: M, E_{K_A}(N_A, K_{ab}), E_{K_B}(N_B, K_{ab})$
  - M4.  $B \rightarrow A: M, E_{K_A}(N_A, K_{ab})$
- Real protocol analysis
  - M1.  $A \rightarrow B: M, A, B, E_{K_A}(N_A, M, A, B)$ 
    - $N_a$  and  $N_b$  are nonces, to prove the freshness of messages.
      - ◆ Since they don't carry any significant information, nonces could be sent in the clear.
        - ◊ In M1, encryption is not for secrecy, but to indissolubly link  $\langle Alice, N_a, M \rangle$  together.
        - ◊ Same thing for M2 for Bob.

This is what happens when nonces are sent in the clear.

**Principle 4.** Properties required to nonces must be clear. What it is fine to guarantee freshness might not be to guarantee an association between parts

**Principles 5.** The reason why encryption is used must be clear

- ! □  $M$  is an identifier of the current instance of the protocol.  
It is created by the initiator of the protocol.
  - ◆  $M$  is a sort of nonce too: so why  $N_a$  and  $N_b$  are not enough?  
 $M$  then disappears in M3 and M4 (more precisely, it's not encrypted, so what is it for then?)
- M2.  $B \rightarrow T: M, A, B, E_{K_A}(N_A, M, A, B), E_{K_B}(N_B, M, A, B)$ 
  - B forwards M1 to the TTP
- M3.  $T \rightarrow B: M, E_{K_A}(N_A, K_{ab}), E_{K_B}(N_B, K_{ab})$ 
  - The TTP generates the key  $K_{ab}$ , which is sent to B.
- M4.  $B \rightarrow A: M, E_{K_A}(N_A, K_{ab})$ 
  - B then forwards the part containing  $K_{ab}$  to A.
- The protocol is structured as a pair of remote procedure calls



- Assumptions

- Shared secret and secret keys

$$A \overset{K_a}{\equiv} A \leftrightarrow T$$

$$B \overset{K_b}{\equiv} A \leftrightarrow T$$

$$T \overset{K_a}{\equiv} A \leftrightarrow T$$

$$T \overset{K_b}{\equiv} A \leftrightarrow T$$

$$T \overset{K_{ab}}{\equiv} A \leftrightarrow B$$

- The last one says that the TTP can establish the shared key  $K_{ab}$  b/w A and B (message M3).

- Freshness

$$A \equiv \#(N_a)$$

$$B \equiv \#(N_b)$$

$$A \models \#(N_a) \quad B \models \#(N_b)$$

$$A \models \#(M)$$

- Set of assumption about trust.

A and B both believe that T is an authority about  $K_{ab}$  (they believe that T is able to generate good shared keys

$$A \models \left( T \Rightarrow A \xleftrightarrow{K} B \right) \quad B \models \left( T \Rightarrow A \xleftrightarrow{K} B \right)$$

- T has the capability to relay

$$A \models (T \Rightarrow B | \sim M) \quad B \models (T \Rightarrow A | \sim M)$$

- In M3, T redirects M both to A and B.

T could change M to M' for B, claiming that A actually sent M' instead of M.

This assumption says that each party has to believe that T has an authority about saying who-sent-what.

- Idealized protocol

- The first message does not change

"messages in plain text have all been removed, because they have nothing to do with the analysis and authentication"

$$\boxed{\text{M1. } A \rightarrow B : \{N_a, M, A, B\}_{K_a}}$$

A says that:

- M is a transaction with B
- $N_a$  is another name of A in M

- Second message

$$\boxed{\text{M2. } B \rightarrow T : \{N_a, M, A, B\}_{K_a}, \{N_b, M, A, B\}_{K_b}}$$

- The TTP believes that the first part of the message comes from Alice  
*it's encrypted with  $K_a$*
- The TTP though doesn't have any proof about the freshness of that part of the message.
- After M2, the TTP knows that:
  - ◆  $T \models A | \sim (Na, M)$
  - ◆  $T \models B | \sim (Nb, M)$

The TTP then knows that A and B are both in the same instance of the protocol (if the triple M, A, and B match).

The TTP has no proof of these message's freshness:

- ◆ M then acts as a global name of the protocol instance
- ◆ Na and Nb act like a local name of the protocol instance.

- Third message

$$\boxed{\text{M3. } T \rightarrow B : \left\{ N_a, A \xleftrightarrow{K_{ab}} B, B | \sim M \right\}_{K_a}, \left\{ N_b, A \xleftrightarrow{K_{ab}} B, A | \sim M \right\}_{K_b}}$$

- First part: as the second part.

- Second part

- ◆ The TTP says to B that  $K_{ab}$  is the session key, and also that "A once said M".

Bob believes that this part comes from the TTP (because Nb proves that this part is fresh (second postulate), so Nb here acts like a nonce).

- Fourth message

$$\boxed{\text{M4. } B \rightarrow A : \left\{ N_a, A \xleftrightarrow{K_{ab}} B, B | \sim M \right\}_{K_a}}$$

- B forwards the first part of M3 to A: nothing new to say.

- Analysis

- After message M2

- Postulate 1a

$$\frac{T \models A \xleftrightarrow{K_a} T, T \lhd \{N_a, M, A, B\}_{K_a}}{T \models A | \sim (N_a, M, A, B)}$$

- Postulate 1a

$$\frac{T \models B \xleftrightarrow{K_b} T, T \lhd \{N_b, M, A, B\}_{K_b}}{T \models B | \sim (N_b, M, A, B)}$$

- After message M3

- Postulate 1a

$$\frac{B| \equiv B \leftrightarrow T, B \lhd \left\{ N_b, A \xrightarrow{K_{ab}} B, B | \sim M \right\}_{K_b}}{B| \equiv T | \sim \left( N_b, A \xrightarrow{K_{ab}} B, B | \sim M \right)}$$

- Postulate 2

$$\frac{B| \equiv \# \left( N_b, A \xrightarrow{K_{ab}} B, B | \sim M \right), B| \equiv T | \sim \left( N_b, A \xrightarrow{K_{ab}} B, B | \sim M \right)}{B| \equiv T | \equiv \left( N_b, A \xrightarrow{K_{ab}} B, B | \sim M \right)}$$

- Postulate 3

$$\frac{B| \equiv T | \equiv A \xrightarrow{K_{ab}} B, B| \equiv T | \Rightarrow A \xrightarrow{K_{ab}} B}{B| \equiv A \xrightarrow{K_{ab}} B}$$

- Postulate 1

$$\frac{B| \equiv (T \Rightarrow A | \sim M), B \lhd \{ A | \sim M \}_{K_b}}{B| \equiv A | \sim M}$$

- After message M4

- The same as the previous one

- Otway-Rees modified

- If nonces had to guarantee freshness only, they could have been sent in the clear:

M1.  $A \rightarrow B : M, A, B, N_A, E_{K_A}(M, A, B)$

M2.  $B \rightarrow T : M, A, B, N_A, E_{K_A}(M, A, B), N_B, E_{K_B}(M, A, B)$

?

- In this way, M1 and M3 (or M2 and M4) are not linked anymore

- ◆ Messages recall

M1.  $A \rightarrow B : M, A, B, E_{K_A}(N_A, M, A, B)$

M2.  $B \rightarrow T : M, A, B, E_{K_A}(N_A, M, A, B), E_{K_B}(N_B, M, A, B)$

M3.  $T \rightarrow B : M, E_{K_A}(N_A, K_{ab}), E_{K_B}(N_B, K_{ab})$

M4.  $B \rightarrow A : M, E_{K_A}(N_A, K_{ab})$

- This is subject to the MITM attack

- ◆ In this example, C impersonates B
- ◆ Suppose C performed an execution of the protocol (called M') with A
- ◆ C stores the part of M2 which is encrypted by

$E_{K_A}(M', A, C)$

- ◆ Suppose C becomes an adversary that wants to act like Bob (in a future connection b/w A and B)
- ◆ C intercepts M1 (so B won't receive it)
- ◆ This is the attack that C can perform:

M1.  $A \rightarrow B[C] : M, A, B, N_A, E_{K_A}(M, A, B)$

M2.  $C \rightarrow T : M', A, C, N_A, E_{K_A}(M', A, C), N_c, E_{K_c}(M', A, C)$

M3.  $T \rightarrow C : M', E_{K_A}(N_A, K_{ab}), E_{K_c}(N_c, K_{ab})$

M4.  $[C]B \rightarrow A : E_{K_A}(N_A, K_{ab})$

- ◊ C can build M2 by using the quantity previously stored.
- ◊ A doesn't know about the M → M' change
- ◊ B doesn't even receive the initiating message
- ◊ The TTP generates M3 because it knows that it's talking to A and C (based on the encrypted messages it received in M2)
- ◊ The protocol now has established a session b/w A and C instead of A and B.
  - ▶ A believes to be in the M protocol execution, but C believes to be in the M' execution.
  - ▶ A thinks s/he's talking to B, while s/he's talking to C instead.

- Countermeasurement

If we need to insert references to Alice and Bob in M3 and M4, then the protocol can be modified as follows

- M1.  $A \rightarrow B: A, B, N_a$
- M2.  $B \rightarrow T: A, B, N_a, N_b$
- M3.  $T \rightarrow B: E_{K_A}(N_a | A, B, K_{ab}), E_{K_B}(N_b | A, B, K_{ab})$
- M4.  $B \rightarrow A: E_{K_A}(N_a | A, B, K_{ab})$

- This is based on principle number 6. This rule should be always followed.  
A message should always be self-contained, and always able to express itself.

**Principle 6.** If an identifier is necessary to complete the meaning of a message, it is prudent to explicitly mention such an identifier in the message

- It is easy to see that C cannot replay that stored quantity anymore.

- SSL (old version)

- The handshake protocol wants to establish a session key b/w the client and the server
- Real protocol

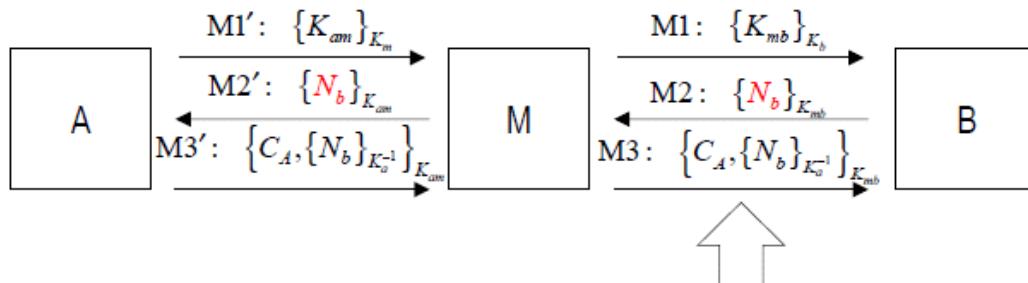
- M1.  $A \rightarrow B: \{K_{ab}\}_{K_b}$
- M2.  $B \rightarrow A: \{N_b\}_{K_{ab}}$
- M3.  $A \rightarrow B: \{C_A, \{N_b\}_{K_a^{-1}}\}_{K_{ab}}$

- Real protocol analysis

- M1.  $A \rightarrow B: \{K_{ab}\}_{K_b}$ 
  - The client encrypts the session key by means of the server's public key.
  - B then sees Kab, but it doesn't know who has sent the message.
  - ◆  $B <| Kab$
- M2.  $B \rightarrow A: \{N_b\}_{K_{ab}}$ 
  - The server (B) says it saw  $K_{ab}$ , and it sends a challenge.
  - $A | = B | \sim N_b$
  - ! □ There's still nothing which links Kab with A (the following attack is then possible).  
B doesn't know who sent the M1 message.
  - This is similar to the exam given July 6th, 2012.
- M3.  $A \rightarrow B: \{C_A, \{N_b\}_{K_a^{-1}}\}_{K_{ab}}$ 
  - $K_a^{-1}$  is A's private key, it is a digital signature.
  - So A in this way responds to the challenge while authenticating itself.
  - A appends its certificate to the message, and encrypts the whole message with the session key.  
The certificate is needed to verify the digital signature.

- Attack

- Examining this protocol with the BAN logic, Bob has no proof that Alice knows Kab.
- Sort of MITM
- Scheme



After M3, Bob believes he is talking to Alice

1. Suppose that A initially wants to connect to M, so A sends M1'.
  2. M sends a message to B, playing the role of the client, so it sends M1.
  3. Now B replies with a challenge (containing a nonce), according to the protocol, by sending M2'.
  4. M receives the nonce and encrypts it by means of Kam, replying to A with M2'.
  5. A responds to the challenge with the last SSL handshake protocol step, with M3'.
  6. M then decrypts M3' by means of Kam and re-encrypts it with Kmb, creating M3 to be sent to B.
- ◆ After M3, B thinks it's talking to A, with the right certificate Ca, with a nonce Nb and with the shared session key

Kmb.

- This attack occurs because there's nothing that links A and the key Kab.

- Solution

- By inserting the nonce  $N_b$  in the digital signature.
- The client is required to digitally sign the nonce, the shared key and the entities.

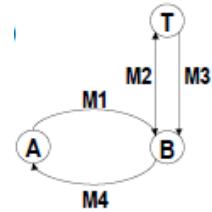
M3 was	and then becomes
M3. $A \rightarrow B : \{C_A, \{N_b\}_{K_a^{-1}}\}_{K_{ab}}$	M3 $A \rightarrow B : \{C_A, \{A, B, K_{ab}, N_b\}_{K_a^{-1}}\}_{K_{ab}}$

- ◆ Signing the nonce just with the key is not sufficient.
  - ◊ It creates a link b/w the session and the shared key.
  - ◊ Protocol modifications (look at the attack)
    - M3' has in its digital signature  $K_{am}$ .
    - M3 has in its digital signature  $K_{mb}$ .
  - ◊ Why identifiers are needed
    - If the adversary selects  $K_{mb} = K_{am}$ , the previous MITM attack would still be possible (because M can forward the digital signature as it is).
    - By inserting the entities IDs in the digital signature, there's no way to forge (forward) the digital signature.
- ◆ That explains principle #7:

**Principle 7.**

- If an entity signs an encrypted message, it is not possible to infer that such an entity knows the message contents
- In contrast, if an entity signs a message and then encrypts it, then it is possible to infer that the entity knows the message contents

- Comparison among real protocols

Needham-Schroeder	$M1 \quad A \rightarrow T \quad A, B, N_a$ $M2 \quad T \rightarrow A \quad E_{K_a}(N_a, B, K_{ab}, E_{K_b}(K_{ab}, A))$ $M3 \quad A \rightarrow B \quad E_{K_b}(K_{ab}, A)$ $M4 \quad B \rightarrow A \quad E_{K_{ab}}(N_b)$ $M5 \quad A \rightarrow B \quad E_{K_{ab}}(N_b - 1)$
Otway-Rees	$M1. \quad A \rightarrow B : \quad A, B, N_a$ $M2. \quad B \rightarrow T : \quad A, B, N_a, N_b$ $M3. \quad T \rightarrow B : \quad E_{K_A}(N_a, A, B, K_{ab}), E_{K_B}(N_b, A, B, K_{ab})$ $M4. \quad B \rightarrow A : \quad E_{K_A}(N_a, A, B, K_{ab})$ 
SSL (old version)	$M2. \quad B \rightarrow A : \quad \{N_b\}_{K_{ab}}$ $M3 \quad A \rightarrow B : \quad \{C_A, \{A, B, K_{ab}, N_b\}_{K_a^{-1}}\}_{K_{ab}}$

### Predictable nonces

- Three ways to build a nonce
  - Timestamp
    - It's very difficult to maintain synchronization in a distributed system.

A replay attack can be performed in this error-window.

- The use of timestamps requires the assumption that authentication has already been assured.
- Nonces may be predictable: timestamps can overcome this problem.

**Principle 8.** A predictable quantity can be used as a nonce in a challenge-response protocol. In such a case, the nonce must be protected by a replay attack

- In this case the predictable quantity is  $N_a$
- Time server example

- ◆ The user A wants to know the current time  $T_s$

◊	$M1$	$A \rightarrow S:$	$A, N_a$
◊	$M2$	$S \rightarrow A:$	$\{T_s, N_a\}_{K_{as}}$

- ◆ Assumptions

- ◊  $A \mid = S \xleftrightarrow{K_{as}} A$
- ◊  $A \mid = S \Rightarrow T_s$
- ◊  $A \mid = \#(N_a)$

Because A generated it.

- ◆ Results obtain after applying the logic

- ◊ Message meaning rule  
 $A \mid = S \mid \sim T_s$
- ◊ Nonce verification rule  
 $A \mid = S \mid = T_s$
- ◊ Jurisdiction rule  
 $A \mid = T_s$

- ◆ Attacks

- ◊ Anybody can act like Alice

**$M$  predicts the next value of  $N_a$**

**$M1 \quad M \rightarrow S \quad A, N_a$**

**$M2 \quad S \rightarrow M \quad \{T_s, N_a\}_r$  ( $S$  receives  $M2$  at time  $T_s$ )**

- ◊ A compromised server sends the wrong timestamp back

- ◆ Solution according to Principle 8 (protecting nonce  $N_a$ )

◊	$M1$	$A \rightarrow S:$	$A, \{N_a\}_{K_{as}}$
◊	$M2$	$S \rightarrow A:$	$\{T_s, \{N_a\}_{K_{as}}\}_{K_{as}}$

- Synchronization

**Principle 9.** If freshness is guaranteed by time stamp, then the difference between the local clock and that of other machines must be largely smaller than the message validity. Furthermore, the clock synchronization mechanisms is part of the Trusted Computing Base (TCB)

- Kerberos example
  - ◆ Server clock set back: authenticators can be reused
  - ◆ Server clock set ahead: it's possible to generate post-dated authenticators
- Counter
  - Assumption: the upper bound has to be very large.
  - It is predictable.
- Random number generator
  - Assumption: it should be impossible to generate the same number twice.
  - It's not predictable.

## GSM protocol

- Client: mobile phone
- Server: the final authentication server (not all the intermediate antennas)
- Real protocol
  - **M1.**  $C \rightarrow S: C$ 
    - The client sends its ID
  - **M2.**  $C \leftarrow S: \rho$ 
    - The server maintains a database <ID, symmetric key contained in the user SIM>

- The server generates a random challenge  $\rho$  and sends it to the client
- The server calculates  $h(K_c, \rho) = < \sigma, K >$ : in this way it generates
  - A challenge  $\sigma$
  - The session key  $K$

o M3.  $C \rightarrow S : \sigma$

- The client sends back the response ( $\sigma$ )

#### Assumptions

$$\begin{array}{l} S \stackrel{K_c}{\equiv} C \leftrightarrow S \quad C \stackrel{K_c}{\equiv} S \leftrightarrow C \\ S \stackrel{\#}{\equiv} \#(\rho) \end{array}$$

#### Idealized protocol

$$M3. \quad C \rightarrow S : \left\langle \overset{K}{C \leftrightarrow S, \rho} \right\rangle_{K_c}$$

#### Results

$$S \stackrel{K}{\equiv} C \stackrel{K}{\equiv} S \leftrightarrow C$$

- Suffers from the chosen-plaintext attack

## Principles

- Idealized messages description

**Principle 1.** We have to specify the meaning of each message; specification must depend on the message contents; it must be possible to write a sentence describing such a meaning

- Assumptions

**Principle 2.** Designer must know the trust relationships upon which the protocol is based. He/she must know why they are necessary. Such reasons must be made explicit.

- Encrypted nonces

**Principle 3.** A key may have been used recently to encrypt a nonce but it may be old or compromised. The recent use of a key does not make it more secure

- The usage of nonces and encryptions must be justified

**Principle 4.** Properties required to nonces must be clear. What it is fine to guarantee freshness might not be to guarantee an association between parts

**Principle 5.** The reason why encryption is used must be clear

- Example: nonces and [encryption](#) in Otway-Rees

- Nonces  $N_a$  and  $N_b$  just prove messages freshness.

- [If nonces had to guarantee freshness only, they could have been sent in the clear.](#)

- Otway-Rees is then modified as following:

$$M1. \quad A \rightarrow B : M, A, B, \textcolor{red}{N_A}, E_{K_A}(M, A, B)$$

$$M2. \quad B \rightarrow T : M, A, B, \textcolor{red}{N_A}, E_{K_A}(M, A, B), \textcolor{red}{N_B}, E_{K_B}(M, A, B)$$

- [M1 and M3 \(or M2 and M4\) are not linked anymore](#)

- ◆ Messages recall

$$M1. \quad A \rightarrow B : M, A, B, E_{K_A}(N_A, M, A, B)$$

$$M2. \quad B \rightarrow T : M, A, B, E_{K_A}(N_A, M, A, B), E_{K_B}(N_B, M, A, B)$$

$$M3. \quad T \rightarrow B : M, E_{K_A}(N_A, K_{ab}), E_{K_B}(N_B, K_{ab})$$

$$M4. \quad B \rightarrow A : M, E_{K_A}(N_A, K_{ab})$$

- Countermeasurement based on [Principle 6](#)

If we need to insert references to Alice and Bob in M3 and M4, then the protocol can be modified as follows

$$M1. \quad A \rightarrow B : A, B, N_a$$

$$M2. \quad B \rightarrow T : A, B, N_a, N_b$$

$$M3. \quad T \rightarrow B : E_{K_A}(N_a, \textcolor{red}{A, B, K_{ab}}), E_{K_B}(N_b, \textcolor{red}{A, B, K_{ab}})$$

$$M4. \quad B \rightarrow A : E_{K_A}(N_a, \textcolor{red}{A, B, K_{ab}})$$

- In M1, encryption is not for secrecy, but to indissolubly link  $< Alice, N_a, M >$  together.

M1.  $A \rightarrow B: M, A, B, E_{K_A}(N_A, M, A, B)$

- Countermeasure to replays or quantity forwarding to avoid impersonation

**Principle 6.** If an identifier is necessary to complete the meaning of a message, it is prudent to explicitly mention such an identifier in the message

- Example: [modified Otway-Rees](#)

If we need to insert references to Alice and Bob in M3 and M4, then the protocol can be modified as follows

M1.  $A \rightarrow B: A, B, N_a$

M2.  $B \rightarrow T: A, B, N_a, N_b$

M3.  $T \rightarrow B: E_{K_A}(N_a, A, B, K_{ab}), E_{K_B}(N_b, A, B, K_{ab})$

M4.  $B \rightarrow A: E_{K_A}(N_a, A, B, K_{ab})$

- Example: [SSL \(old version\) with  \$K\_{am} = K\_{bm}\$](#)

M3  $A \rightarrow B: \{C_A, \{A, B, K_{ab}, N_b\}_{K_a^{-1}}\}_{K_{ab}}$

- If the adversary selects  $K_{mb} = K_{am}$ , the MITM attack would still be possible (because M can forward the digital signature as it is).
- By inserting the entities IDs in the digital signature, there's no way to forge (forward) the digital signature.

- Signing encrypted data

**Principle 7.**

- If an entity signs an encrypted message, it is not possible to infer that such an entity knows the message contents
- In contrast, if an entity signs a message and then encrypts it, then it is possible to infer that the entity knows the message contents

- Example: X.509

$A \rightarrow B: A, \{T_a, N_a, B, X_a, \{Y_a\}_{K_b}\}_{K_a^{-1}}$

The message contains no proof that the sender (Alice) knows  $Y_a$

- A could simply include  $\{Y_a\}_{K_b}$  in its digital signature, without knowing  $Y_a$ , because it is encrypted by means of  $K_b$ .

- Example: [SSL \(old version\)](#)

M3  $A \rightarrow B: \{C_A, \{A, B, K_{ab}, N_b\}_{K_a^{-1}}\}_{K_{ab}}$

- The entity (A) first signs the message and then encrypts it.

- Predictable nonces should be protected (encrypted)

**Principle 8.** A predictable quantity can be used as a nonce in a challenge-response protocol. In such a case, the nonce must be protected by a replay attack

- In [time servers](#), instead of:

▪ M1	$A \rightarrow S:$	$A, N_a$
▪ M2	$S \rightarrow A:$	$\{T_s, N_a\}_{K_{as}}$

This could be done:

▪ M1	$A \rightarrow S:$	$A, \{N_a\}_{K_{as}}$
▪ M2	$S \rightarrow A:$	$\{T_s, \{N_a\}_{K_{as}}\}_{K_{as}}$

- Timestamps synchronization

**Principle 9.** If freshness is guaranteed by time stamp, then the difference between the local clock and that of other machines must be largely smaller than the message validity. Furthermore, the clock synchronization mechanisms is part of the Trusted Computing Base (TCB)

- Example: [compromised Kerberos server's clock](#) brings some problems.

- Self-contained messages

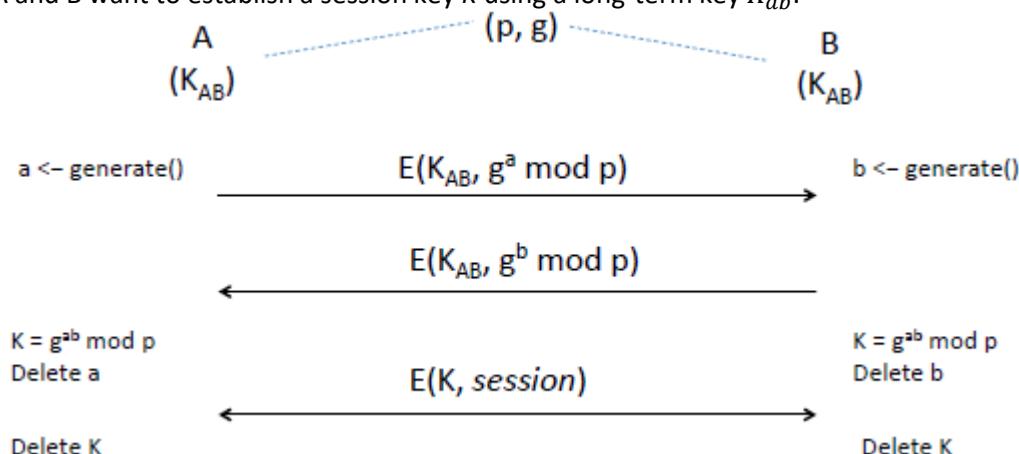
**Principle 10.** The contents of a message must allow us to determine:  
(i) the protocol the message belongs to, (ii) the execution instance of  
the protocol, (iii) the number of the message within the protocol

- Example: in Needham-Schroeder, [the response has  \$N\_b - 1\$](#) , just to distinguish this message from the challenge.

# Perfect forward secrecy

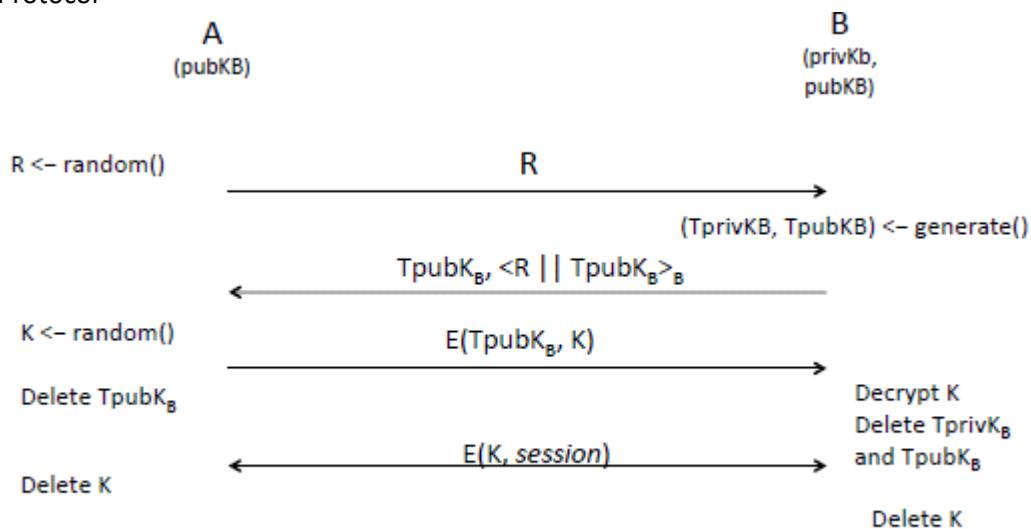
martedì 23 maggio 2017 11:07

- Intro
  - A and B have already their pair of public and private keys
  - A generates a symmetric secret K randomly and sends it to B by encrypting it with the public key of the latter.
  - At the end of the session, both parties have to delete the shared secret session key.
- **Problems with the compromization of a long-term secret (such as B's private key)**
  - Every future session involving B is compromised.
    - Nothing to do other than revoking B's keys and warning A.
  - **Also past sessions are compromised** if the adversary was able to sniff the initial ciphertext containing the shared key.  
If B's private key is compromised, the adversary is then able to recover K from the ciphertext and hence to read all past sessions.
    - Perfect forward secrecy is about this.
      - **(DEF) PFS:** Disclosure of long-term secret keying material does not compromise the secrecy of the exchanged keys from earlier runs
      - PKE, and in particular DH, makes it possible to achieve this requirement
- Pre-Shared Key Ephemeral Diffie-Hellman (PSK-DHE)
  - Summary: it just uses DH for every session to establish a different session key K everytime. Once the DH random numbers  $a$  and  $b$  are deleted, nobody is able to decrypt past communications.
  - Assumptions
    - $A| \equiv A \xleftrightarrow{K_{ab}} B$
    - $B| \equiv A \xleftrightarrow{K_{ab}} B$
  - The long-term secret in this case is the shared key  $K_{ab}$ , like IPsec (a pre-shared key is installed manually on router pairs in order to let them calculate other session keys).  
A and B want to establish a session key K using a long-term key  $K_{ab}$ .



- Perfect forward secrecy in this case wants to assure that previous communications cannot be read if the shared key is compromised
- A encrypts the public key  $|g^a|_p$  with the shared key  $K_{ab}$  to guarantee authenticity
  - It's used to avoid MITM attacks

- There's no need to encrypt public keys
- A and B, according to DH, are able to generate the  $K$
- Secrets  $a$  and  $b$  (and the corresponding public quantities) can be deleted
  - $a$  and  $b$  are generated on-the-fly just once during the session
- Suppose that  $K_{ab}$  has been compromised and that an adversary has sniffed all messages
  - The adversary has still to solve the discrete-logarithm problem
  - Past sessions are protected
- Summary
  - Pre-Shared Key Ephemeral Diffie-Hellman
  - Ephemeral Diffie-Hellman
    - Keys  $a$  and  $b$  are ephemeral (one-time per-session or per message)
  - Once  $a$  and  $b$  (and  $K$ ) have been deleted there is no way to recover  $K$ , and thus the session, even if the long-term private  $K_{ab}$  is compromised: neither A nor B can
- Costs
  - The DH protocol has to be performed for each session, and it is computationally heavy.
    - Generate two large numbers.
    - Several exponentiations.
- Ephemeral RSA (RSAE)
  - Protocol



$\langle x \rangle_B$ : Bob's digital signature on  $x$ ; attach  $Cert_B$  is necessary

- A sends a random quantity  $R$  to B.

- B has to generate a public and private key pair on-the-fly

- The ephemeral B's public key is signed by B.

**B also appends its certificates** so A can really trust B.

(it is shown as  $\langle \rangle_B$ )

The random quantity is attached to prove freshness.

- B's public key compromised:

- In order to decrypt previous sessions, the adversary has to calculate  $K$ , but in order to do this it has to have B's private key.

B's private key gets deleted as soon as possible though, as defined by the protocol.

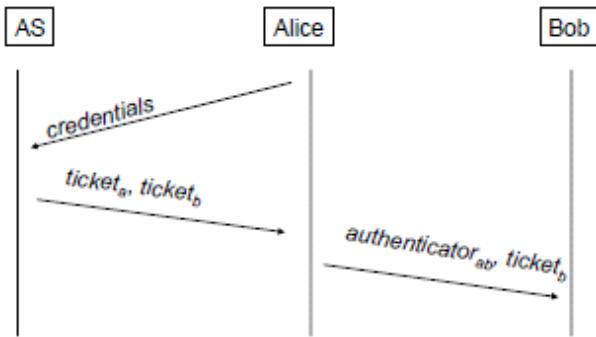
# Kerberos

martedì 23 maggio 2017 11:04

- Kerberos is a computer network **authentication protocol** that works on the basis of 'tickets' to allow nodes communicating over a non-secure network to prove their identity to one another in a secure manner
- Based on the [Needham-Schroeder protocol](#)
- **Kerberos** (or KDC) is a TTP (AS + its DB + TGS) b/w clients and servers
- Provides
  - ★ Mutual authentication b/w client and server
  - ★ Key establishment b/w client and server
  - ★ Prove that the client is active and vice versa
- Entities

```
graph LR; AS[Authentication Server] -- Kc --> Alice[Alice (client)]; AS -- Ks --> Bob[Bob (server)]; Alice <--?--> Bob;
```

*$K_c$  e  $K_s$  (master key) sono segreti condivisi tra AS e client e server, rispettivamente (ad esempio derivati da password)*
- A: client (workstation or user)
  - Shares  $K_a$  with the AS
  - Shares  $K_{ab}$  with B
- B: server
  - Shares  $K_b$  with the AS
  - Shares  $K_{ab}$  with A
- AS: authentication server, the TTP
- Past architecture on which Kerberos was based
  - More workstations (PCs) connected to a DCE (Distributed Computer Environment)
  - The DCE is connected to an AS (Authentication Server) and to a FS (File Server)
  - A PC had to authenticate itself to the AS
  - Every workstation then had its own home directory on the FS, and PCs cached their own directory
    - That's why those clients were called thin clients, because they didn't store any data
- Every party has a shared key with each other entity.
  - Those keys are related to users, not to machines.
    - This allows users to login to different computers.
  - ! ○  $K_a = f(P_a)$ 
    - Keys are functions of passwords: A, with its password  $P_a$ , can generate  $K_a$ .
- Requirements
  - Security
    - Secrecy
    - Authenticity
  - Availability
    - If the AS is down, no user is able to work.
    - Replication, uninterruptable power systems, ...
  - Transparency
    - A user just has to type his/her password
  - Scalability
- Basic idea



- Messages

$M1$	$A \rightarrow AS$	$A, B, t, L, N_a, WS_A$
$M2$	$AS \rightarrow A$	$\{A, B, t, L, K_{ab}, WS_A\}_{K_b}, \{B, t, L, N_a, K_{ab}\}_{K_a}$
$M3$	$A \rightarrow B$	$\{A, B, t, L, K_{ab}, WS_A\}_{K_b}, \{A, t_a, subkey_a\}_{K_{ab}}$
$M4$	$B \rightarrow A$	$\{t_a, subkey_a\}_{K_{ab}}$

- Messages detailed analysis

- M1 and M2 are used by A to login to the AS (to get tickets), exchanged just once during one session (L validity).

▪ $M1$	$A \rightarrow AS$	$A, B, t, L, N_a, WS_A$
--------	--------------------	-------------------------

- $t$  is a timestamp, the **login time**
- $L$  is the session time length (**ticket validity interval**, usually a **couple of hours**)
- $N_a$  is a nonce
- $WS$  is A's workstation address

▪ $M2$	$AS \rightarrow A$	$\{A, B, t, L, K_{ab}, WS_A\}_{K_b}$	$\{B, t, L, N_a, K_{ab}\}_{K_a}$
		$ticket_b$	$ticket_a$

- The AS creates the session key  $K_{ab}$
- These two quantities are called **tickets**,  $ticket_b$  and  $ticket_a$ .
  - ◆ They are for **key authentication**.

- M3 and M4 are exchanged everytime that A wants to use B's service (using the tickets).

▪ $M3$	$A \rightarrow B$	$\{A, B, t, L, K_{ab}, WS_A\}_{K_b}$	$\{A, t_a, subkey_a\}_{K_{ab}}$
		$ticket_b$	$authenticator_{ab}$

- In the first part, A forwards  $ticket_b$  to B in order to inform the latter about the session key  $K_{ab}$ 
  - ◆ This is kind of a certificate.
  - ◆ WS is included in the first part to specify that  $K_{ab}$  is valid when A is using the machine identified by WS.
- The second part of the message is called the **authenticator**
  - ◆ This is the **key confirmation** message for B according to the BAN logic
    - ! ◆ "A" is the known quantity that confirms the key
  - ◆ Encrypted by means of  $K_{ab}$
  - ◆ It contains another timestamp,  $t_a > t$ , the **time in which A wants to use B's service**
  - ◆  $subkey_a$  can be used for the actual fulfillment of the service
    - ◆ All the future session communication (connections?) can be encrypted with different subkeys, instead of  $K_{ab}$ .
    - ! ★  **$K_{ab}$  is then used just for authentication, meanwhile the subkeys are used to confidentiality.**
    - ◆ Subkeys can be computed from  $K_{ab}$ , like SSL does.

$M4$	$B \rightarrow A$	$\{t_a, subkey_a\}_{K_{ab}}$
------	-------------------	------------------------------

Key confirmation for A

**!**   $t_a$  should be a known quantity that confirms the key.

- Ban logic analysis

- Assumptions

- Starting from pre-shared keys

$$A \stackrel{K_a}{\equiv} A \leftrightarrow AS \quad B \stackrel{K_b}{\equiv} B \leftrightarrow AS$$

$$AS \stackrel{K_a}{\equiv} A \leftrightarrow AS \quad AS \stackrel{K_b}{\equiv} B \leftrightarrow AS$$

$K_a$  and  $K_b$  derive from users' password.

$K_a = f(password_a)$ , where the function could be anything (an hash, for example).

- The TTP generates the shared key between A and B

$$AS \stackrel{K_{ab}}{\equiv} A \leftrightarrow B$$

- Both A and B believe that the TTP is an authority on shared keys

$$A \stackrel{K_{ab}}{\equiv} \left( AS \Rightarrow A \leftrightarrow B \right) \quad B \stackrel{K_{ab}}{\equiv} \left( AS \Rightarrow A \leftrightarrow B \right)$$

- When B receives the first part of M3, it thinks that it comes from the AS.

Has B any proof that that message is fresh?

The timestamp was generated by A (it was a [problem of the Needham-Schroeder protocol](#)).

This also applies for the second part of M3:  $t_A$  was generated by A.

An important assumption is:

$$A \stackrel{\#}{\equiv} (t)$$

$$B \stackrel{\#}{\equiv} (t) \quad B \stackrel{\#}{\equiv} (t_A)$$

**!** In practice, it means that **Kerberos requires synchronized clocks** (between A and B).

For A, it's easy to believe that t and  $t_A$  (omitted on the slides) are fresh quantities, because they're generated from her.

Problem:

◆ If an adversary:

- ◇ Has an old key and its related ticket
- ◇ Succeeds in turning back the clock

Then it can reuse the key

◆ Solution: using certificates in M1 (so there won't be the need of shared secrets based on reusable passwords)

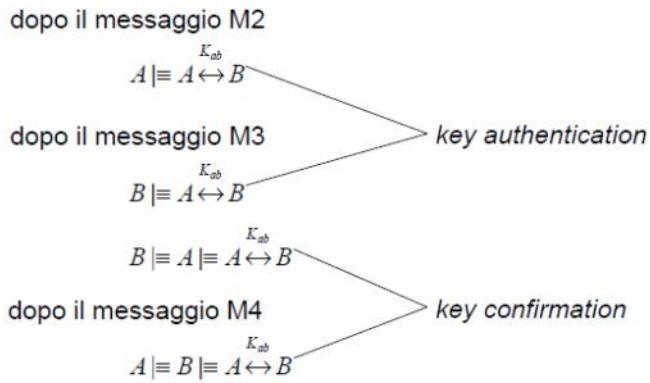
#### Procedure PKINIT

$$M1. \quad A \rightarrow T \quad S_A(A, B, N_A), certificate_A$$

$$M2. \quad T \rightarrow A \quad ticket_B, E_{e_A}(S_T(K, N_A, L, B))$$

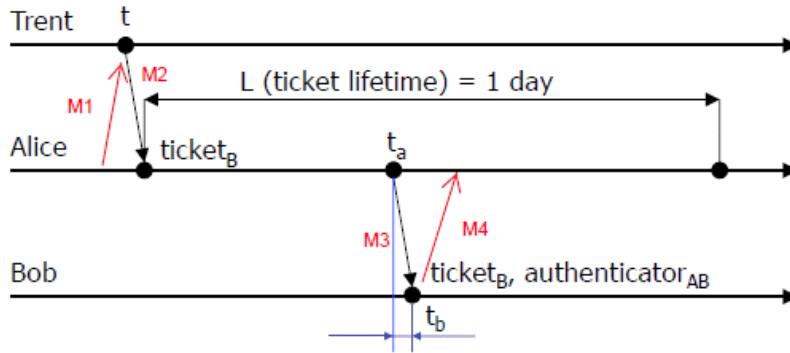
Alice holds  $certificate_T$

- Analysis



- **Athenticators and tickets lifetime**

- Scheme



$|t_b - t_a| \leq \lambda$  (authenticator lifetime)  
 authenticator lifetime  $\approx$  clock skew  
 $(\lambda = 5 \text{ minuti})$

- Whenever L expires, the user A has to re-type her password.
- $t_a$  is not instant: M3 is received by B after a certain delay
  - There's no specific range window: by increasing its size, a **replay attack** is more likely to occur.
  - By replaying the authenticator, B accepts the subkey again, so all the previous messages are re-usable.
  - By making it too short, it may happen very frequently that the authenticator is no longer considered fresh.  
This affects usability, causing a **Denial of Service**.
  - Its typical length (the **authenticator validity**) it's around a **few minutes**.

- Summary

- Requires synchronized clocks
- $K_a$  and  $K_b$  derive from users' password, so they are as secure as the latter.
- The AS should be highly available: server replication.
  - Better have a stateless Authentication Server.

- Complete architecture

- Kerberos was developed to, for example, allow access to shared File Servers.

For other services like mail, all those messages have to be exchanged every time.

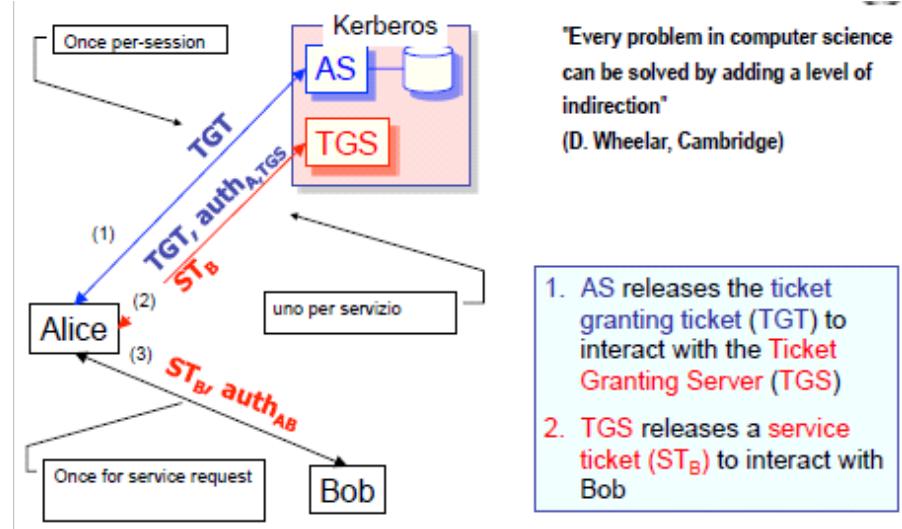
- ! ○ **Usability problem:** When the ticket expires, the user has to re-type his/her password.

- Solutions

- A user's WS (WorkStation) could store (cache) the password for the whole session in order to avoid repeating M1 and M2.
    - This is not considered secure enough because the password is a single point of failure.

- **Ticket Granting Service (TGS)**

- Service implemented on the AS.
    - In order to interact with this service, a ticket **TGT** (Ticket Granting Ticket) is needed.
    - **The TGS issues tickets for other services.**



1. Alice interacts with the AS (M1 and M2) and receives a TGT, a ticket (lasts hours, whole session) to use with the TGS

- ◊ The AS establishes **TK** (Ticketing Key) as the shared key between Alice and the TGS.
- ◊ Messages

$M1$	$A \rightarrow AS$	$A, TGS, t, L, N_a, WS_A$
$M2$	$AS \rightarrow A$	$\{A, TGS, t, L, TK, WS_A\}_{K_{TGS}}, \{TGS, t, L, N_a, TK\}_{K_a}$

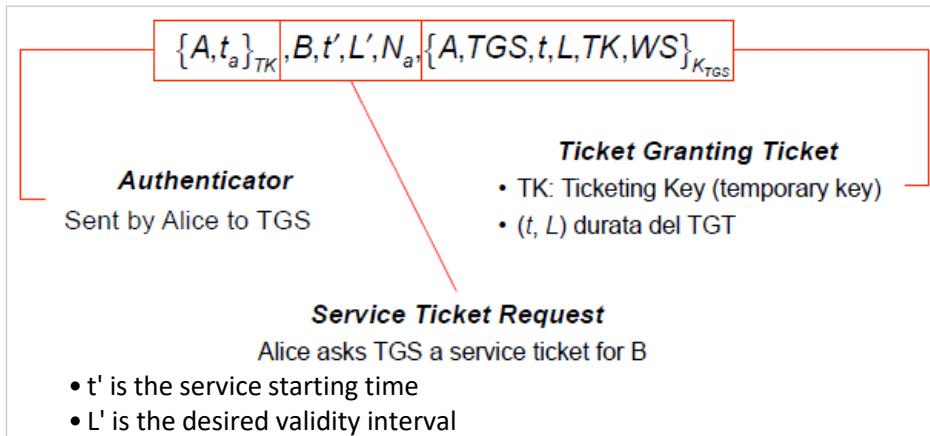
2. By means of the TGT, the user can interact with the TGS using an authenticator.

i)  $A \rightarrow TGS$

► Composition

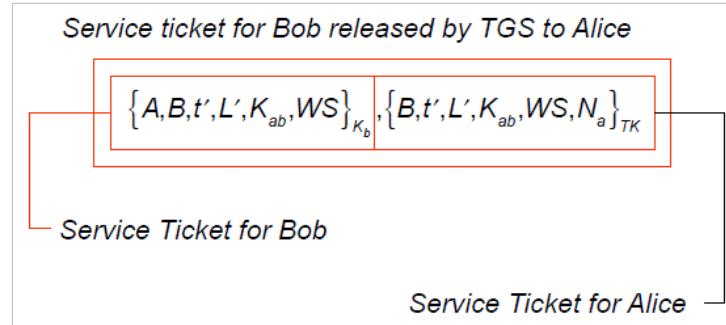
- It's mainly an M1 message  $(A, B, t', L', N_a, WS_A)$ 
  - ◊ The Service Ticket **Request** part
  - ◊ "Mainly" because the TGS will respond with an M2 message containing 2 tickets.
- M3 message  $(\{A, TGS, t, L, TK, WS_A\}_{K_{TGS}}, \{A, t_a\}_{K_{ab}})$ 
  - ◊ With this message, the client authenticate itself and presents the TGT that will allow him to interact with the TGS
  - ◊ Usually authenticators also have a subkey in it, but in this case is useless since the client doesn't have to establish a key for encrypting future communications with the TGS
  - ◊  $t'$  is the service starting time
  - ◊  $L'$  is the desired validity interval

► Scheme



ii)  $TGS \rightarrow A$

- ▶ The TGS can then release **service tickets** (like  $ST_b$ ) to interact with other services (in this case the server Bob).
- ▶ This is an M2 message.



- ◊ The interaction with the TGS is the same as the interaction of any other service.
  - a. Alice uses  $ST_b$  (the Service Ticket for Bob) to interact with the server Bob.
- Improvement: the user has to type his/her password just **once**.  
The TGT is stored in the user's WS and his/her password can be then deleted.  
Whenever a user has to interact with another service, s/he can use the TGT.
  - ◆ This introduces a limit of the amount of time during which an adversary can impersonificate a user.
    - ◊ The TGT can still be compromised, but the TGT does not last forever (a ticket has its own validity  $L'$ ).
- M3 and M4 have to be exchanged with the service
  - ◆ M3
    - ◊ To inform B about the established session key  $K_{ab}$
    - ◊ To perform **key confirmation** to B thanks to the authenticator (encrypts "A", a known data, with the  $K_{ab}$  key)
  - ◆ M4
    - ◊ To let B confirm the established session key  $K_{ab}$

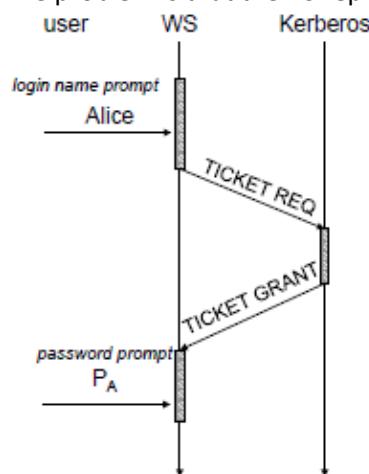
- Problems

- Those problems exist because Kerberos was developed during the 80s, where wireless communication was not spread like nowdays.

Hence eavesdropping was not considered as a problem.

1. Kerberos does not authenticate users w.r.t. the AS

- The AS, once it receives M1, doesn't have any clue if A is really A.
- **Offline password attack**: an adversary may send M1 on Alice behalf.  
The AS replies with M2, so the adversary can use A's ticket to launch a password attack, by guessing password and verifying them by decrypting the ticket.
- The problem is that the AS replies with M2 before checking for the user's password

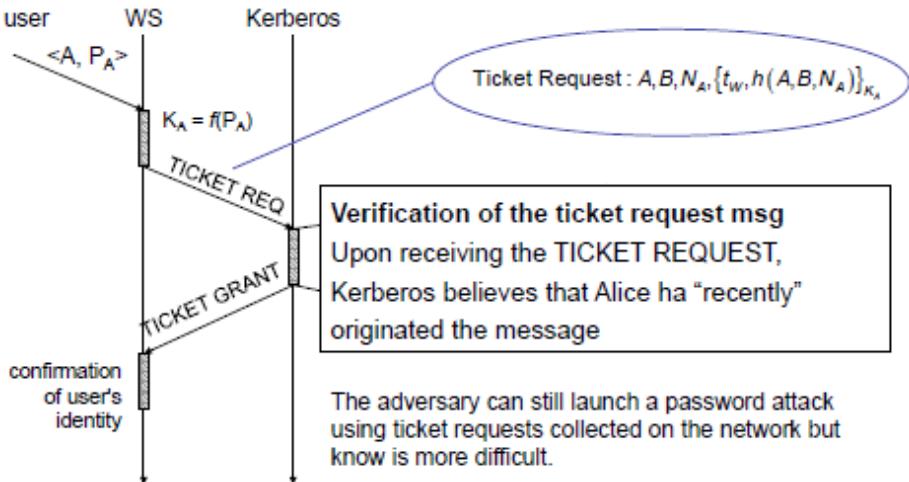


- WS does not authenticate Alice but the ticket is encrypted.
- However, an adversary can collect tickets (on demand) and use them to launch a pwd attack (known plaintext attack)

- The AS, once it has received M1, has no clue if the message was really originated by Alice.
- By replying with M2 (TICKET\_GRANT), it can be stored in order to perform an **offline password**

**attack.**

- Solution: M1 has a new format



- It is not in the clear anymore: it includes a part encrypted with  $K_a$ .
  - ◆ In this way, the user has to type his/her password **at the beginning** (in order to generate  $K_a$ )
- In this way the AS can check if the M1 message was generated by Alice or not.
  - ◆ If the authenticity check fails, the AS doesn't reply with M2, so the adversary is not able to perform an offline attack.
    - ◊ The adversary could still perform an **online** attack, but it's more difficult.  
It could be discouraged by increasing the waiting interval once a user fails to login.
  - ◆ Otherwise, M2 it's enough for the user to say that s/he's been logged in successfully.

- Mailing service delegation

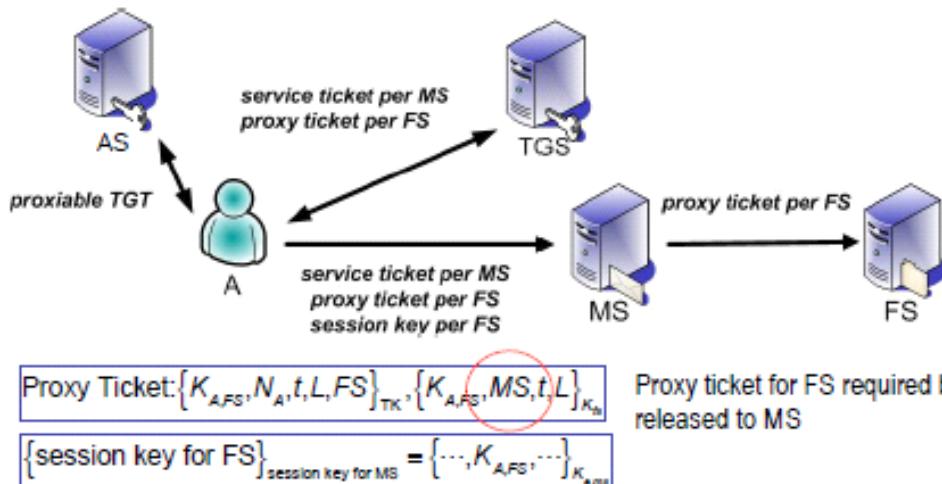
- What's the problem

- The Mail Server (MS) has to interact with the File Server (FS) to save emails (typically on files).  
Of course this job of interacting with the FS is delegated to the MS, so the user doesn't have do to this.
    - ! ▪ The **MS has to be able to write in different directories on the FS**, one for each user.

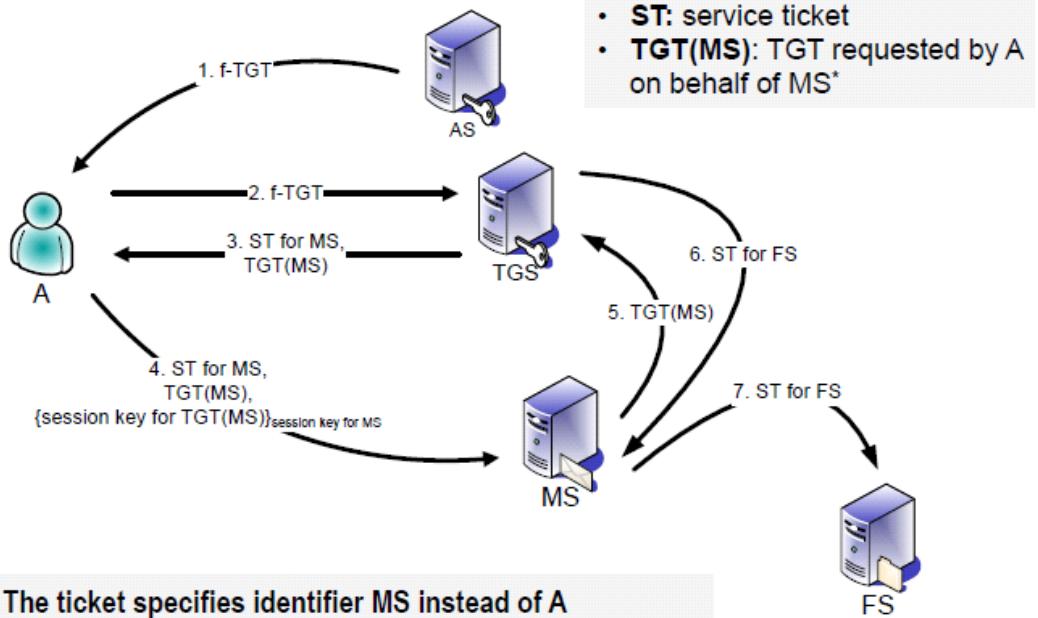
- Solutions

- Root privileges can be given to the MS (like in Sendmail).
      - The MS is now a single point of failure.
    - Solutions (proxy ticket and forwardable TGTs) based on the *principle of minimum privilege*: security policy that basically says that privileges have to be as strict as possible everytime to avoid the previous problem.
      - Proxy tickets

*PT allows us to request a service ticket linked to an address (WS) different from the requesting one*



- a. The user A exchange M1 and M2 with the AS, requiring a TGT with a particular bit set to 1 (PROXIABLE\_TGT := 1).  
Let's call it **PTGT** (Proxiable TGT).
- b. A asks the TGS for a **service ticket** to talk with MS ( $ST_{MS}$ ).
  - ◊ It contains a key that allows A to talk with the MS
$$ST_{MS} = \{ \dots, K_{A,MS}, WS_A, \dots \}_{K_{MS}}, \{ \dots, K_{A,MS}, WS_A, \dots \}_{TK}$$
  - ▶ (In M2) One ticket is encrypted with MS' key, and the other is encrypted with the key shared between A and the TGS (which is TK).
  - ▶ The ticket for the server also contains the address of the user's current workstation (to avoid replay attacks).
- c. There's an extra step: the proxy ticket for the FS.  
A **also** asks the TGS for a special service ticket (**proxy ticket**) which **will allow the MS to interact with the FS**.  
**A will forward this ticket to the MS: the latter will then use this special proxy ticket to access the FS file system.**
  - ◊ This ticket will be used by MS to the FS, delegated by the user A.
  - ◊ Since it is a ticket to talk with the FS, this ticket will contain the key to talk with the FS:
$$\text{Proxy Ticket for FS} = \{ \dots, K_{A,FS}, WS_{MS}, \dots \}_{K_{FS}}, \{ \dots, K_{A,FS}, WS_{MS}, \dots \}_{TK}$$
  - ▶ A normal service ticket would include  $WS_A$ .
  - ▶ The proxy ticket instead includes  $WS_{MS}$ .
    - If it would contain  $WS_A$ , the FS would only accept service requests coming from A, and it would refuse any connection from the FS.
  - ▶ When the FS is gonna receive this proxy ticket, it will allow the MS to access its file system.
  - ▶ In order to let the MS talk to the FS, it needs  $K_{A,FS}$ .  
The only one that can send this key to the MS is the user A.
    - The user A then also has to send the session key  $\{K_{A,FS}\}_{K_{A,MS}}$  encrypted it with the shared key b/w A and the MS,  $K_{A,MS}$ , which is included in the service ticket.
- ◆ Properties
  - ◊ This protocol can be made even more difficult by assigning to the MS only particular privileges (*read only*, ...).
  - ◊ The delegation lasts for the validity ticket L.
    - ▶ So the MS can abuse only during this time interval.
- ◆ Drawback: at login, the user **A has to ask all tickets** (and proxies tickets) s/he needs.  
The client should then know in advance what to ask.  
This could be a (software engineering) problem because:
  - ◊ A client has to have the knowledge on how this protocol is implemented
  - ◊ Any change in implementation affects users behaviour.
- Forwardable TGT
  - ◆ In the proxy tickets solution, the ticket allowed delegation: now, A asks for a forwardable TGT in order to send it to the MS, so the latter can ask all the service tickets it needs.
    - ◊ This solves the previous software engineering problem in the proxy tickets Kerberos version.
    - ◊ It is then now more dangerous from a security point of view, because now compromising the MS could bring more damage than before.
  - ◆ Scheme



- [...]
- Step 3.** TGS returns Alice
  - A **service ticket** for MS  
 $\{..., K_{a,ms}, ...\}_{TK}, \{..., K_{a,ms}, ...\}_{Kms}$
  - A **TGT** containing the **ticketing key** associated to MS instead of Alice  
 $\{..., TK', ...\}_{Ka}, \{..., TK', MS, ...\}_{Ktgs}$
- Step 4.** Alice forwards the two tickets to MS together with the ticketing key **TK'** encrypted by means of  $K_{a,ms}$
- [...]

The ticketing key is associated to MS instead of A

#### Key summary

- TK: shared key b/w A and the TGS
- Tk': shared key b/w the MS and the TGS

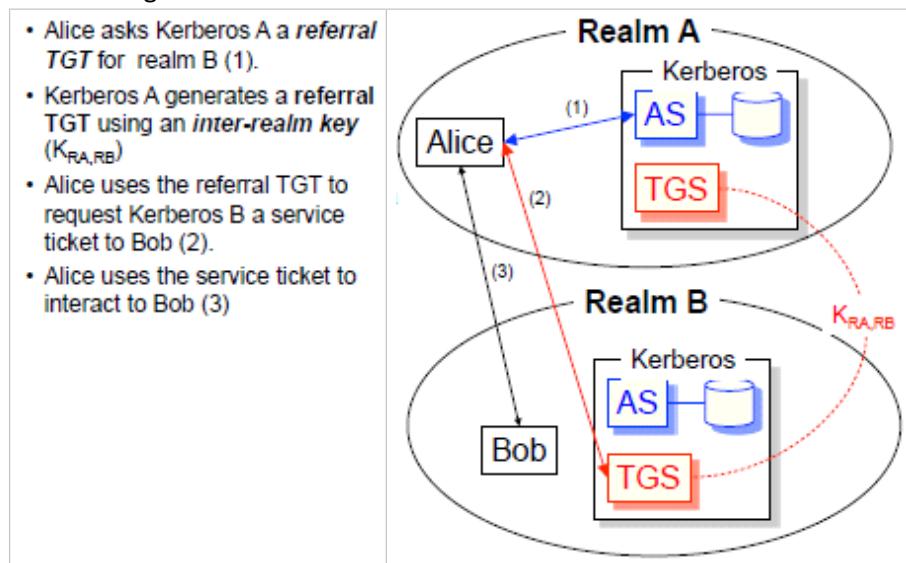
#### □ Comparison between Proxy Tickets and Forwardable TGTs

	Proxy ticket	Forwardable ticket
<b>Pro</b>	The user controls which rights to delegate the server. <i>The user requests the proxy ticket for the FS.</i>	The server determines which ticket it needs. <i>The user does nothing.</i>
<b>Con</b>	The user needs to know which tickets will be necessary. <i>The user requests the proxy ticket for the FS.</i>	A compromised server can abuse of all rights. <i>The server requests all the tickets it needs.</i>

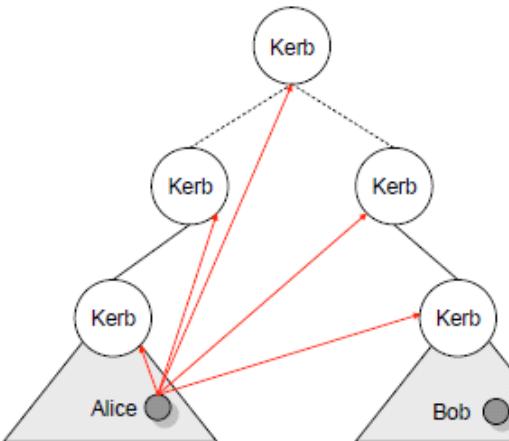
- Limitations: a ticket
  - Has a maximum lifetime
  - Specifies a maximum number of access rights
- Realms and referral tickets
  - **Realms** are administrative domains.
  - Kerberos always authenticates users in its realm.

- Referral TGTs with inter-realm keys
- Functioning

- Alice asks Kerberos A a *referral TGT* for realm B (1).
- Kerberos A generates a referral TGT using an *inter-realm key* ( $K_{RA,RB}$ )
- Alice uses the referral TGT to request Kerberos B a service ticket to Bob (2).
- Alice uses the service ticket to interact to Bob (3)



□ Realms hierarchy



- What happens if something gets compromised
  - Workstation: damages are limited to the WS and its users
    - ◆ A user only requests tickets.
  - Server: damages are extended to all server's users
    - ◆ Good practice: distribute a server
    - ◆ A server
  - Kerberos (or KDC): system completely broken
    - ◆ It stores all the users' secret keys (for user machines and services) in its database
    - ◆ It generates shared keys between users
- Clock synchronization attacks
  - Server clock set back: authenticators can be reused
  - Server clock set ahead: it's possible to generate post-dated authenticators