Summary of Computer Security

Intro

Security Properties:

- **Confidentiality**: is the unauthorized reading of files; ;
- **Integrity**: insure that the data has not been altered (tampered);
- **Authentication**: verification of identity of a person or system. Methods for authentication can be:
  - smartcard
  - password
  - Fingerprint or biometric details
- **Availability**: insure that the data or service is accessible when desired;
- **Accountability**: insure that an illegal action can be tracked down to the responsable;

Privacy vs Secrecy:

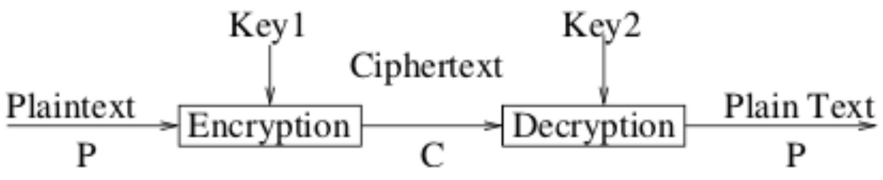
Privacy is for **individuals** and for anonymity;

Secrecy pertains to confidentiality for **organization**;

Cryptography

There are 2 types: **symmetric** and **asymmetric** (or public key) cryptography.

A general schema is:



Here the security depends on the **secrecy of the key** not the algorithm.

Symmetric:

They use the same key to encrypt and decrypt. They can ‘process’ the message by fixed block size or by bit (or byte) at the time.

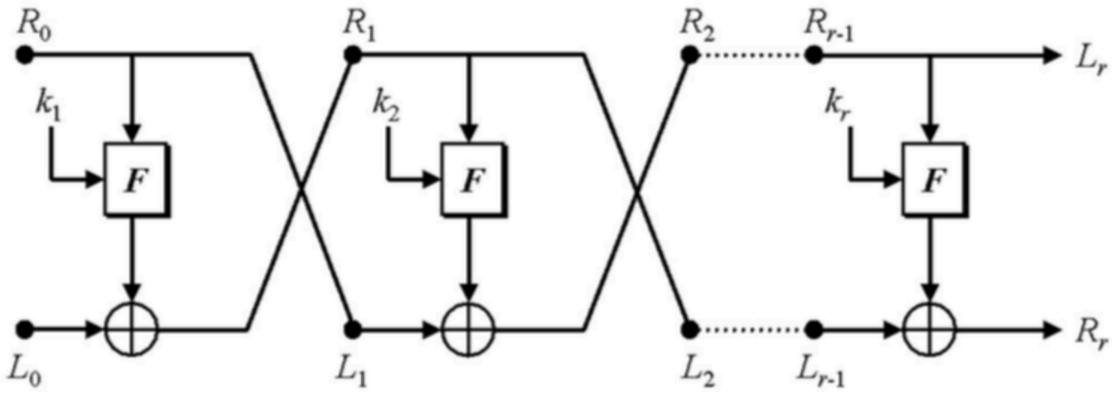
The problem with symmetric schemas is that we need a secure way to share a common secret key.

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.

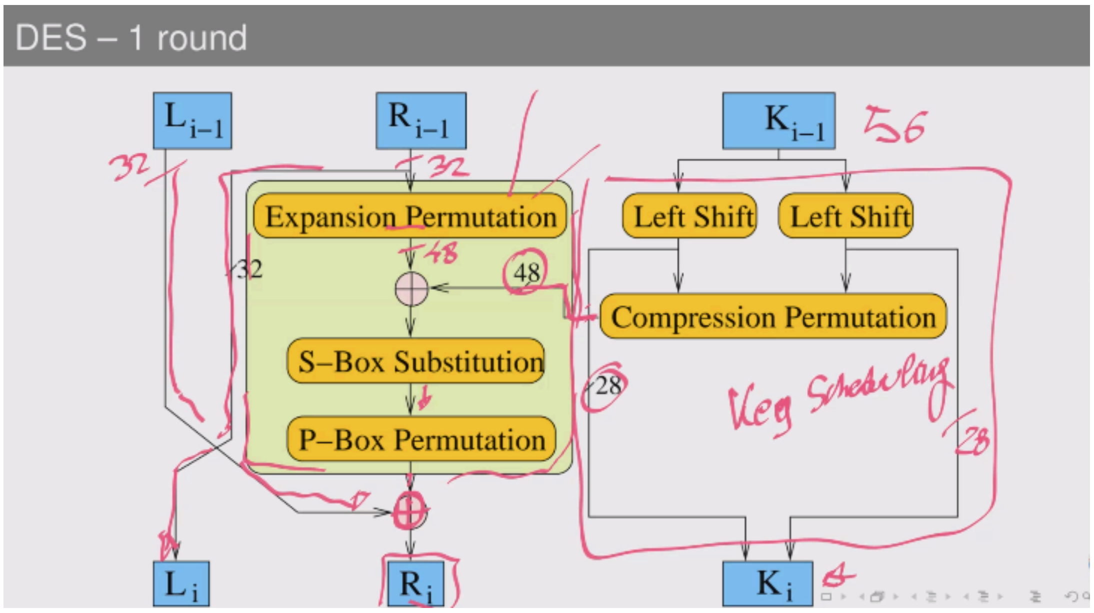
Feistel Substitution - Permutation Cipher

SP nets are based on **confusion** & **diffusion** of message and key meaning that cipher needs to completely obscure statistical properties of original message.



Encryption and Decryption are structural identical.

DES: Data Encryption Standard



It's a 64 bit block cipher. It uses a 56 bit key (wich has 8 bit more for parity check) meaning that it has 2^{56}= 7.2 x 10^{16} possible values.

It perfroms:

1. an initial permutation
2. 16 rounds of Feistel cipher and key-schedule
3. a last inverse permutation

There are different types of DES:

- Single DES
- Double DES: perfrom 2 encryptions with 2 different keys (total: 112 bit key). It's not really more secure than Single DES
- Triple DES:
    - Classic: uses 2 keys to perform 3 different encryption. Is retro compatible with single DES if k1==k2
    - Three-key 3DES: 3 different keys for a total of 168bit key. Used in PGP and S/MIME.

AES: Advanced Encryption Standard

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.

Asymmetric or Public Key

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.

When **encrypting**, you use their public key to write a message and they use their private key to read it.

When **signing**, you use your private key to write message's signature, and they use your public key to check if it's really yours.

With public key cryptography we are able to achieve both *secrecy & authentication* but it does not fully solve the key exchange problem.

RSA Algorithm

We assume that sender and receiver both know the same number (n) in bits calculated as the moltiplication of 2 prime numbers. It splits the plaintext it in chuncks of $log_2(n)$ bits.

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) where:
  - $d = e^{-1} * mod \Phi(n)$
  - $\Phi(n) = (p-1)(q-1)$

The algorithm is divded in 3 parts: generation of keys, encryption and decryption.

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 $\cdots$ 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 = ...

The key distribution problem:

We see 2 different approches:

- **Secret key distribution with RSA:**

  Not too secure, if the private key (d,n) gets compromised, then *k* can be recovered.

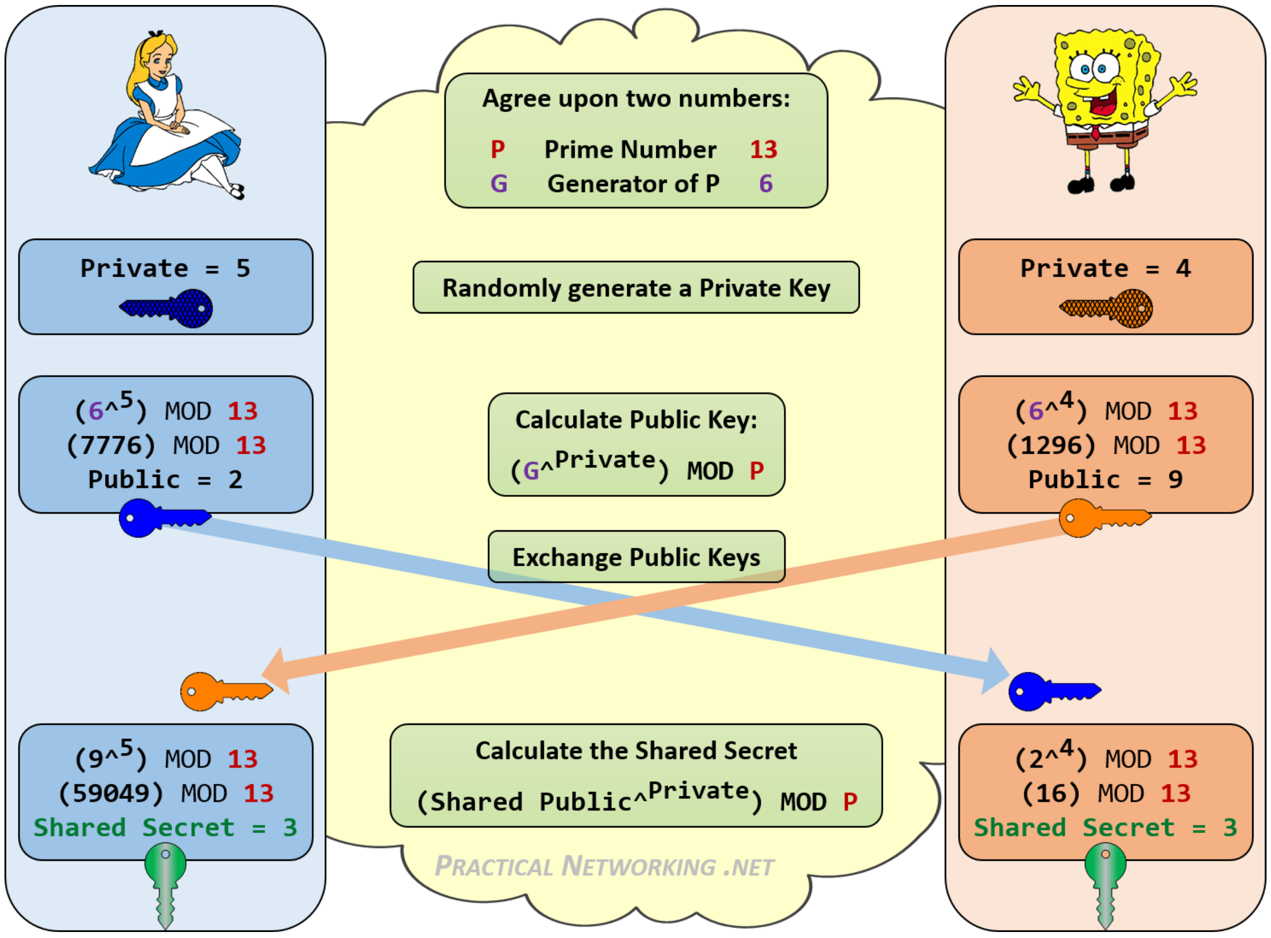
  Encryption of *m* (with public key (*e*, *n*)):

  - 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 $m = D_k(c_2)$
- **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.



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.

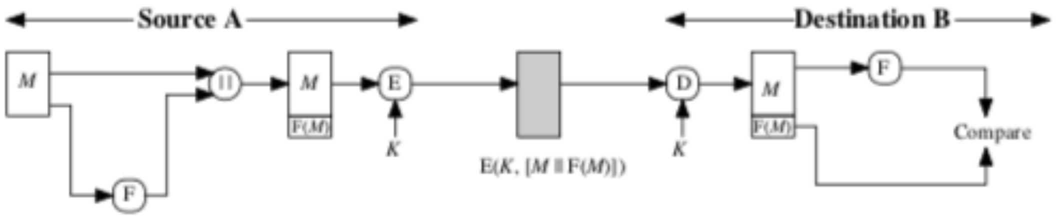
Message Authentication and Digital Signature

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. We append a checksum to the message before encryption.

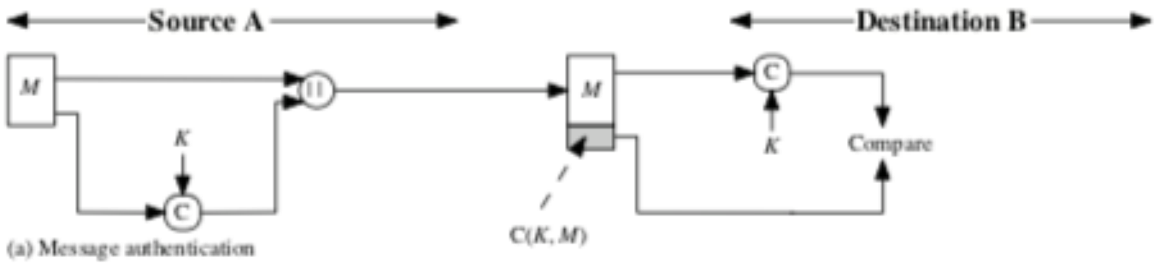


(a) Internal error control

Message Authentication Code (MAC)

The MAC value protects a message's data integrity, as well as its authenticity, by allowing verifiers to detect any changes to the message content.

MAC is a function that taken a message (M) and a secret key (K), returns a fixed-size output: C(M,K). This “tag” is then used by the receiver to check integrity and authenticity.



(a) Message authentication

Cryptographic Hash Function:

Is an algorithm that takes an arbitrary amount of data input (like 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 (in a database peraphs) instead of the password itself, and later used to verify the user.

Hash functions do NOT uses keys, and it’s purpose is to produce a “fingerprint” of a file (or data). To be useful an hash function must follow some properties like: it must be a **one-way function** and must be easy to compute $H(x)$ for any given x.

Digital Signature

A **digital signature** is a scheme for **verifying the authenticity** of digital messages or documents. It 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

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: We feed 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.

There are 2 types of digital signatures: direct (it only includes two parties. Less secure) and arbitrated (there are three parties: sender, receiver and arbiter. More secure thanks to a timestamp added by default.)

Encryption vs Signing:

Encryption is a two-way function; what is encrypted can be decrypted with the proper key.

Hashing is a one-way function that scrambles plain text to produce a unique message digest. With a properly designed algorithm, there is no way to reverse the hashing process to reveal the original password.

Security Protocols

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]}

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.

- Goals: authenticate messages, binding them to their originator, ensure timeliness (recent, fresh) and guarantee the secrecy of certain items.

Needham-Schroeder Public Key Authentication Protocol (with Lowe's Fix)

The goal is to reach a mutual authentication.

1. A —> B : $\{A, N_A\}_{K\_B}$ = Alice send to Bob his 'name' and a nonce
2. B —> A : $\{N_A, N_B, \mathbf{B}\}_{K\_A}$ = 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

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:

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

It suffers from type-flow attacks.

Andrew Secure RPC protocol

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

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

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**.)

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

It suffers form man-in-the-middle attacks.

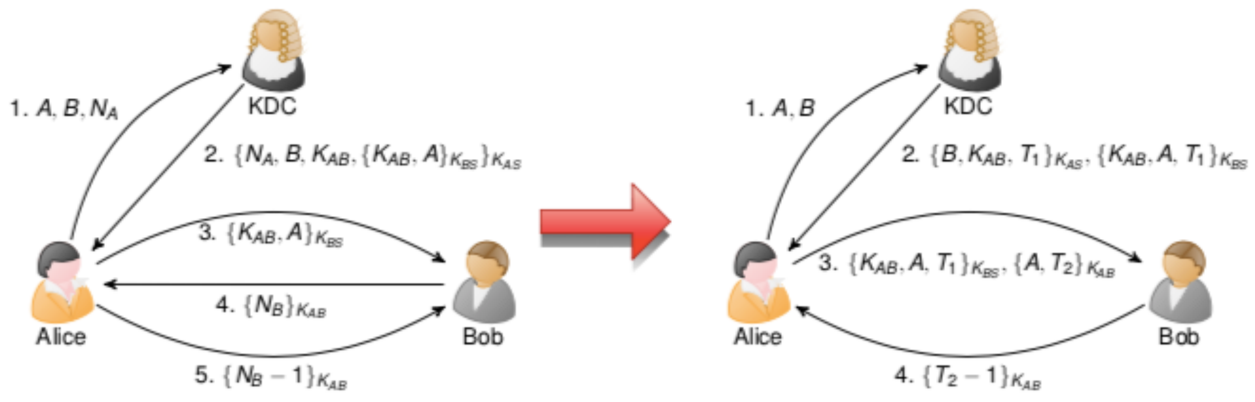
Kerberos

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.

The user's password never travel over the network (and is never stored anywhere), it supports single sign-in, it's easy for admins to disable accounts…

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


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

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 **$K_{A,TGS}$** 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.

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

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

> 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

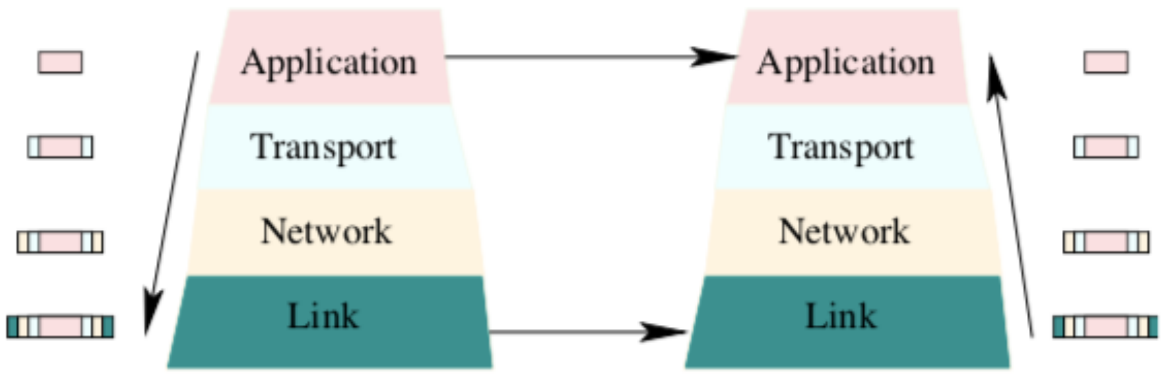
$$\text{5. A} \rightarrow \text{B}: \quad \underbrace{\{A,\ B,\ K_{AB},\ T_3\}_{K_{BS}}}_{\textit{ServTicket}},\ \underbrace{\{A, T_4\}_{K_{AB}}}_{\textit{authenticator}}$$

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

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

Internet Security

Layers



- Application: where the app run. We have services like: telnet, ftp, http, smtp …
- transport (and 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

Where to inject security?

- **Aplication layer** using software like Kerberos. But if we need to modify the software in order to use security, it might have a huge cost.

- **Between Application and Transport layer** using SSL. (We only secure the application we want to use)

  [+] 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

- **Transport layer** using IPSec: we need to modify the OS but all the applications will be secured.

  [+] Transport layer security without modifying applications

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

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

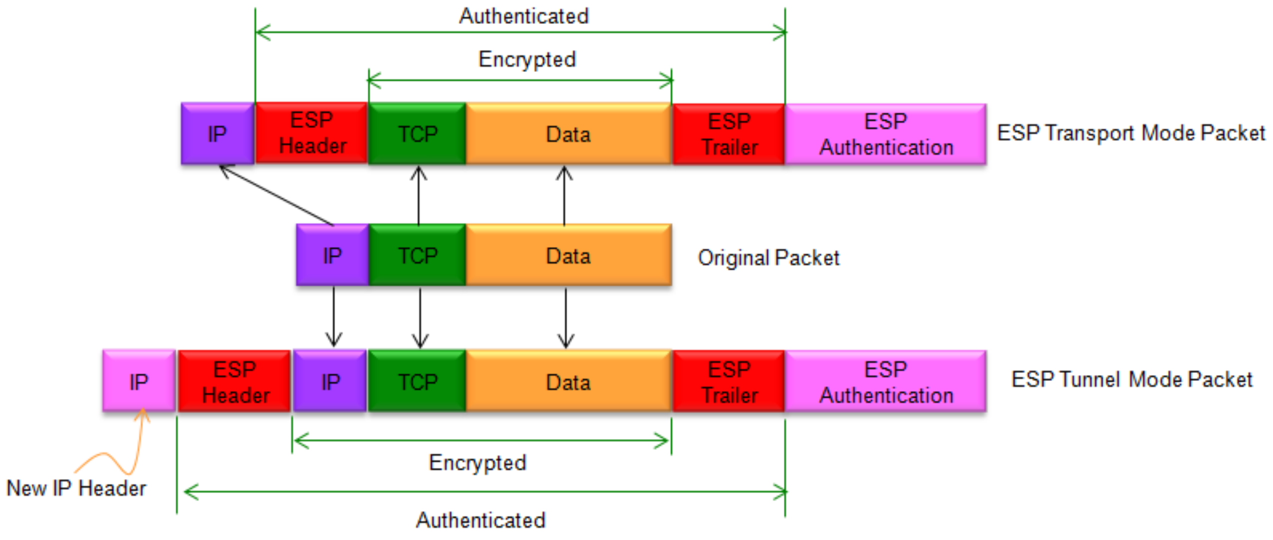
IPSec

Secure channel for all application between 2 hosts.

It provides: confidentiality and authentication.

It is possible to install both on network devices (routers) and computers. So even if i don't have a secure network device i can still use this technology installing it on my pc.

IPSec can be used in transport mode or tunnel mode. The main difference between the two is that in tunnel mode we obfuscate the original IP address.



There are different way to implement IPSec we will see: authentication header (AH) and Encapsulating Security Payload (ESP) (the one in the picture above).

Authentication Header mode (AH)

it’s an extra header added to the packet containing:

- **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

Encapsulating Security Payload mode (ESP)

It adds data 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.

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		✓	✓

Access Control:

A Security Policy is a document that specifies *who can do what*. Example: " Armando can edit 'file1' ".

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.

DAC

It uses Access Control Matrix, Access Control List (like unix) and Capabilities List to keep track of privileges of subjects on objects.

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)

Harrison-Ruzzo-Ullman Model

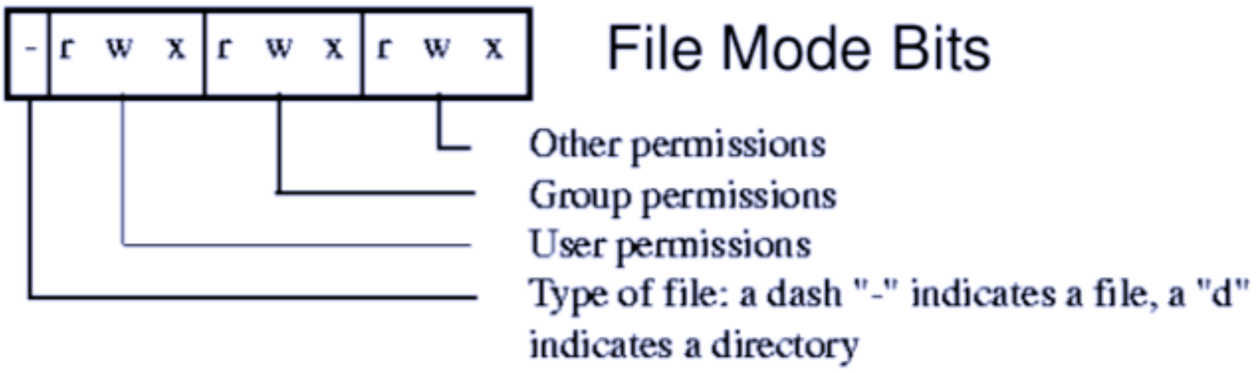
Is a language that allow to specify the policy by using the access control matrix model. Examples:

```
/* Si scriva il comando per create(s, o), con il quale s viene creata la
* risorsa o e viene dato ad s il solo diritto di possesso (Own).
*/

command create(s,o)
  create object o
  enter Own into M(s,o)
end
```

```
/* Si scriva il comando per transfer.write(s1, s2, o), con il quale se s1
* ha il diritto di scrittura su o allora lo trasferisce (perdendolo) a s2.
*/

command trasfer.write(s1,s2,o)
  if Write in M(s1,o)
  then enter Write into M(s2,o)
        delete Write from M(s1,o)
end
```

Access Control List 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`.

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.

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.

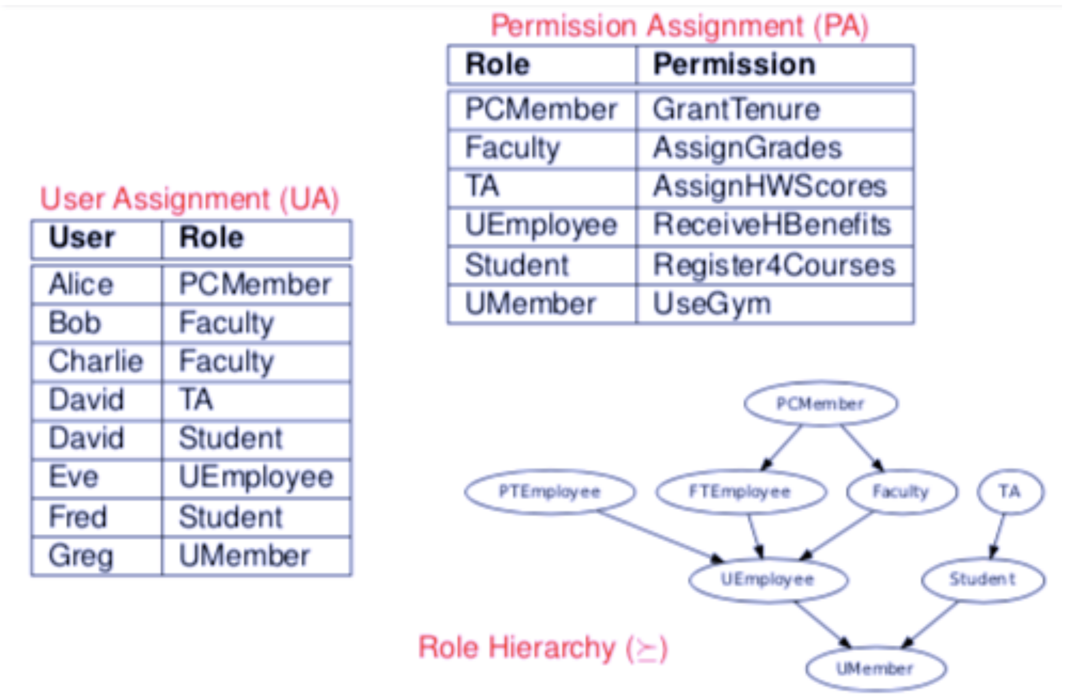
There are 4 different models of MAC:

- the Bell-La Padula model: NoReadUp and NoWriteDown properties.
- the Biba model: Realx NoReadDown (it allow a subject to read down, but first lower subject integrity level to that of the object being read) and Relax NoWriteUp (lower object level to that of subject)
- the Chinese Wall model: the focus is in avoiding **information flow that causes a conflict of interest** between Companies. There are Company datasets and conflict classes
- the Clark-Wilson model (integrity of transaction)

RBAC

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).



Permission Assignment (PA)

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

User Assignment (UA)

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

To update user’s role we can use:

```
//Un user UEmployee può dare permessi di PTEmployee ad uno Student se e solo se
non è anche TA
can_assign: UEmployee: {Student,¬TA} ==> PTEmployee

//Un user Uemployee può revocare i permessi di uno Student segnandolo come "not
a student" (so the account get only UMember benefits)
can_revoke: UEmployee: {Student} ==> ¬Student
```

Security Elements

Smartcards: useful to digitally sign documents. They have a mini processor inside that is used to encrypt document' hashes with a **private key (that never leaves the card itself)**. The smart card then send back to the computer a digitally signed hash of the doument. 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.

Buffer Overflow

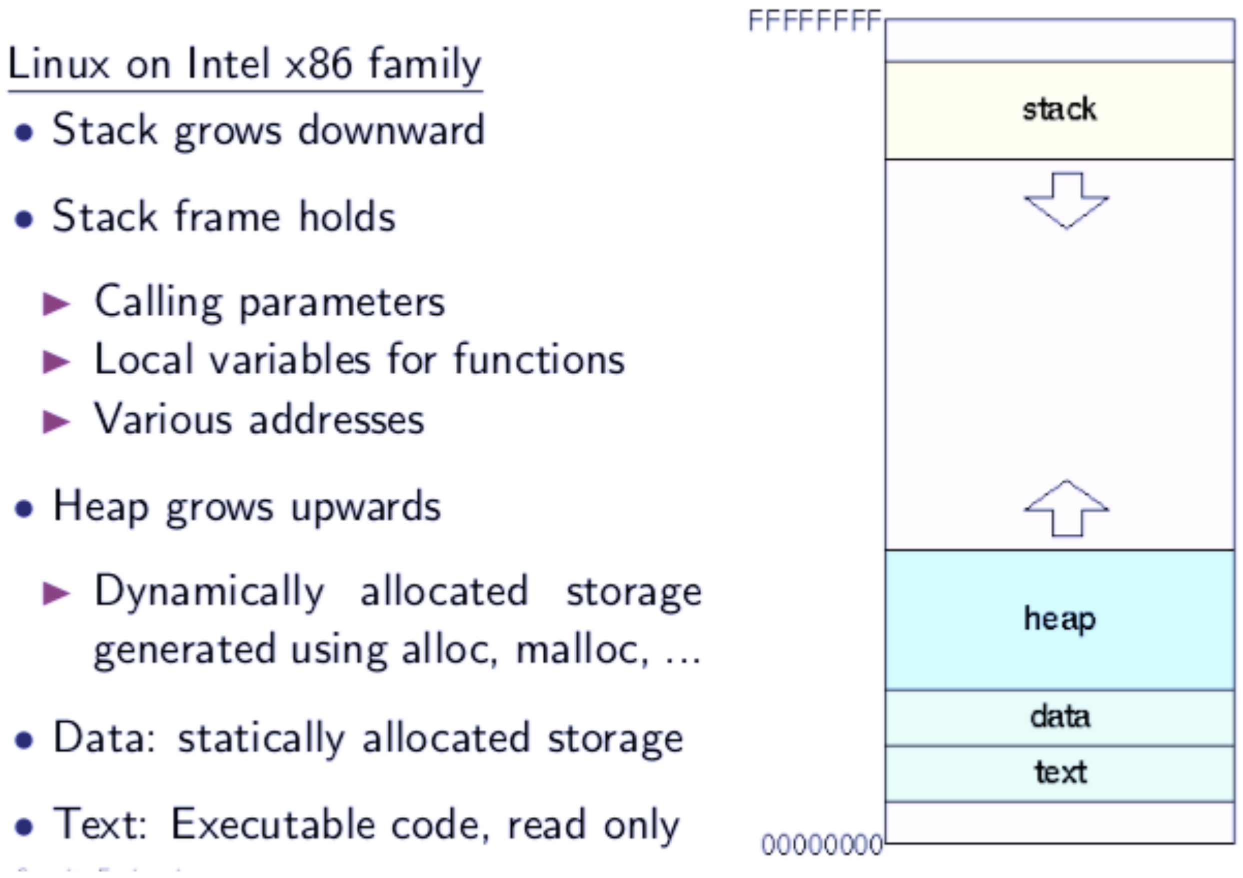
Buffer: a contiguous region of memory storing data of the same type. Used by program to store data.

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

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 address of a memory cell; `*ptr` is the content of the memory cell; `&i` is the address of the cell storing 'i'; `&f` is the address of a function 'f';

Arrays : are contiguous memory cell.

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

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

Esercizi

Symmetric cryptography:

[3luglio2008]

> Si consideri l'algoritmo di cifratura a blocchi definito da:
>
> C0 = IV Ci = EK(Ci−1)⊕Pi (1)
>
> dove IV e' il vettore di inizializzazione. Si definisca l'operazione di decifrazione.

se moltiplichiamo ambo il lati di (1) per EK(Ci−1) otteniamo:

EK (Ci−1) ⊕ Ci = EK (Ci−1) ⊕ (EK (Ci−1) ⊕ Pi)

Sfruttando le proprietà dello ⊕ otteniamo:

EK(Ci−1)⊕Ci = Pi

[20giugno2006]

> Si determini il plaintext corrispondente al ciphertext "dcbdmoet" ottenuto applicando la procedura di cifratura di Vigegnère ed utilizzando come chiave la sequenza di numeri 4 2 11. Si consideri l'alfabeto italiano.

Siccome K = {4,2,11} so che devo dividere il ciphertext in blocchi da 3. Usiamo l'alfabeto italiano quindi 21 lettere. Per calcolare il plaintext è sufficiente "sottrarre" k al cipher text:

dcb dmo et

d-4 c-2 b-11 d-4 m-2 o-11 e-4 t-2

zan zib ar

[14giugno2007]

> Si assuma di avere una conoscenza parziale del seguente plaintext e del corrispondente cipher- text:
>
> Plaintext: + U A + + I A I + + E R I + + E
>
> Ciphertext: I + E C F A + + O H + C KS P +
>
> dove + indica i caratteri non noti.
>
> Si sa che il ciphertext è ottenuto dal plaintext applicando l'algoritmo di Vigenère con lunghezza del blocco pari a 4 e utilizzando l'alfabeto inglese (ovvero ABCDEFGHIJKLMNOPQRSTUVWXYZ). Si determini la chiave.

Alfabeto inglese quindi 26 chars. Per trovare le chiavi, mi baso su ciò che mi viene fornito. Ad esempio nel primo quartetto plaintext: "+ U A +" e cipher: "I + E C" alla 3a posizione sappiamo sia il plain che il cipher. Quindi:

$e(A) = (A + k3) \bmod(26) = E$;

$e(A) = (1 + k3) \bmod(26) = 5 \longrightarrow k3 = 4$

nello stesso modo calcolo k1, k2 e k4

$K1 = 2$; $K2 = 18$; $K3 = 4$; $K4 = 11$

Public Key cryptography

[cs-esercizi-esame]

> Un sistema crittografico a chiave multipla è caratterizzato da un insieme di n chiavi K = {K1,...,Kn} tali che se se C0 = P è un generico plaintext e Ci+1 = E(Ci,Ki) per i = 0,...,n−1, allora Cn =P.
>
> Ovvero cifrando P con tutte le chiavi K1,...,Kn (in qualunque ordine) si ottiene il plaintext di partenza.
>
> (a) Un sistema crittografico a chiave multipla con n = 2 corrisponde ad uno dei sistemi crittografici visti a lezione. Quale? Si giustifichi la risposta data.

(a) Uno schema crittografico a chiave a chiave pubblica (ad esempio RSA), dove K e\` dato dalla chiave pubblica e dalla chiave privata.

> (b) Si discutano i possibili utilizzi di un sistema crittografico a chiave multipla con n > 2.

(b) Potrebbe essere utile nella **firma digiatale congiunta di documenti** tra 2 o più agenti. Ad esempio se avessimo 3 agenti e 4 chiavi, ognuno cifra con la sua chiave e manda al prossimo fino a quando tutti hanno "firmato". Per controllare la firma basterà prendere l'output finale e cifrarlo con la 4a chiave (pubblica).

[10luglio2009]

> (a) Dire quali dei seguenti oggetti digitali puo\` essere utilizzato come certificato digitale per un sito web

(a) un documento contenente <u>il nome del sito, la sua chiave pubblica, il tutto cifrato con la chiave privata di un'Autorità di Certificazione.</u>

> (b) Alice deve inviare un file M di grosse dimensioni a Bob (ad esempio un filmato di qualche Gbyte) in modo tale che Bob sia certo dell'integrità della trasmissione. Quali tra le seguenti procedure sono adeguate allo scopo?

(b) Alice calcola ed invia a Bob: M e la fingerprint di M cifrata con la propria chiave privata.

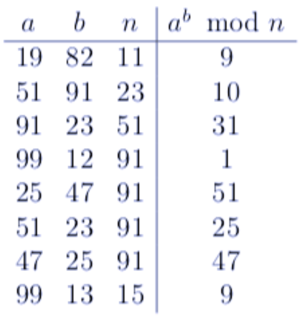
> (c) Supponete di essere proprietari di una azienda e che un vostro cliente vi invii un contratto firmato digitalmente in cui si impegna a pagare una fornitura di beni da voi prodotti ad una somma stabilita a 120 giorni dalla consegna dei beni stessi. La verifica della firma digitale va fatta:

(c) Prima di effettuare la fornitura.

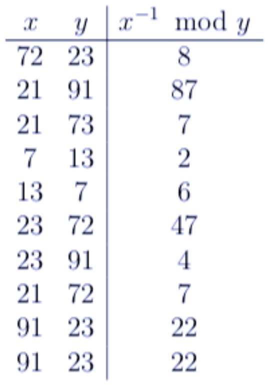
PKC: RSA

[14giugno2007]

> Si consideri l’algoritmo RSA con `p = 7, q = 13 e = 23.`
>
> (a) Si calcoli il testo cifrato C corrispondente al testo in chiaro M = 51. (b) Si calcoli la chiave di decifrazione (d, n). (c) Si verifichi che decifrando C con (d, n) si riottiene M .
>
> Si giustifichino le risposte date scrivendo tutti i calcoli intermedi. Si utilizzino le seguenti tabulazioni (ovviamente parziali) delle funzioni di esponenziazione modulare e dell’inverso moltiplicativo modulo n:



a	b	n	$a^b \bmod n$
19	82	11	9
51	91	23	10
91	23	51	31
99	12	91	1
25	47	91	51
51	23	91	25
47	25	91	47
99	13	15	9



x	y	$x^{-1} \bmod y$
72	23	8
21	91	87
21	73	7
7	13	2
13	7	6
23	72	47
23	91	4
21	72	7
91	23	22
91	23	22

0. n = $p * q$ = 7*13 = 91.

Ricordiamo che le funzioni di cifratura (C) e decifratura (D) si calcolano come:

C = $M^e\ mod(n)$ E = (e,n) = (23,91)

D = $C^d\ mod(n)$ F= (d,n) = (?,91)

1. C = $M^e mod(n) = 51^23mod(91) = 25$

Per calcolare la chiave di decrifrazione ‘d’ bisogna conoscere $\Phi(n) = |Phi(91) = (p-1)(q-1) = 6*12 = 72$

2. Calcolo (d,n)

   d = $e^{-1}mod\Phi(n) = 23^-1mod72 = 47$

3. Decifra C con (d,n)

   D = $C^d mod(n) = 25^{47}mod(91) = 51$

PKC: Diffie-Hellman

[20giugno2006]

> Alice manda a Bob il numero 5 e Bob manda ad Alice il numero 8. I valori del modulo e del generatore α sono (pubblicamente) noti essere rispet- tivamente pari a q = 11 e α = 2. Si determini la chiave condivisa da Alice e Bob alla fine dell’esecuzione del protocollo.

Sappiamo che:

- Ya = $\alpha^{X_a} mod(q) = 2^{X_a} mod(11)$ = 5 (ciò che Alice manda a Bob)
- Yb = $\alpha^{X_b} mod(q) = 2^{X_b} mod(11) = 8$ (ciò che Bob manda ad Alice)

Calcolo X_a = 4 e $X_b = 3$

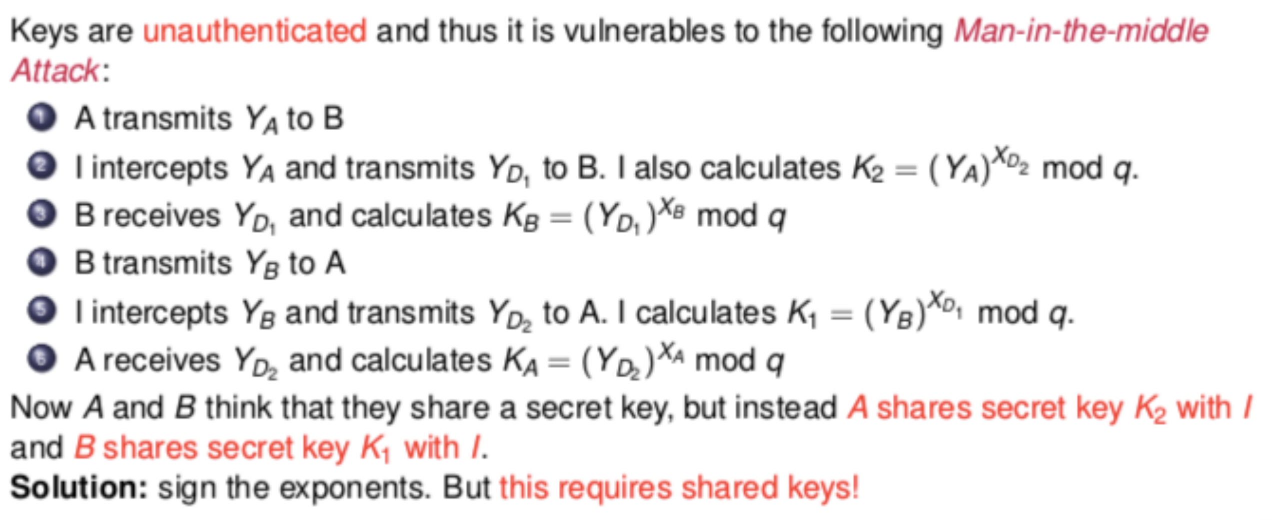
Posso ora trovare la chiave condivisa finale: $K_A = Y_b^{X_a} mod(q) = 8^4 mod(11) = 4096 mod 11 = 4$

$K_b = Y_a^{X_b} mod(q) = 5^3 mod(11) = 125 mod(11) = 4$

Dato che coincidono (4) va bene.

[12giugno2008]

> Si dimostri che il protocollo per scambio di chiavi di Diffie-Hellman non garantisce l'autenticazione.


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!

[10luglio2009]

> Si consideri lo schema di Diffie-Hellman dove q = 11 e α = 2.
>
> (a) Se A ha chiave pubblica YA = 9, qual è la chiave privata di A?
>
> (b) Se B ha chiave pubblica YB = 3, qual è la chiave segreta K?

(a) Ya = $\alpha^{X_a} mod(q) = 2^{X_a} mod(11)$ = 9 ——> Xa = 6 (tentativi, altrimenti dovrei usare il logaritmo per "abbassare" l'esponenziale e viene un casino)

(b) 'K' segreta = $Y_a^{X_b} mod(q) = 9^{X_b} mod(11)$

calcolo X_b sapendo che Yb = $\alpha^{X_b} mod(q) = 2^{X_b} mod(11) = 3$ quindi $X_b = 8$

K = $9^8 mod(11) = 3$

Sec Protocols

[14giugno2007]

> Si consideri il seguente protocollo P1 per la creazione di una chiave condivisa tra due agenti
>
> A e B:
>
> 1. A→B:{A,Na}Kb
> 2. B→A:{B,Nb}Ka
> 3. A → B : {zero,Msg}Na⊕Nb
> 4. B → A : {one,Msg}Na⊕Nb
>
> dove zero e one sono identificatori distinti e Na ⊕ Nb è lo XOR bit a bit di Na e Nb.
>
> (a) Si descrivano i singoli passi del protocollo e le proprietà di sicurezza per il quale è stato presumibilmente progettato.

Il protocollo dovrebbe garantire:

1. la mutua autenticazione tra A e B
2. lo scambio confidenziale di una nuova chiave Na⊕Nb e di Msg.

> (b) Si discuta se il protocollo garantisce o meno le proprietà di sicurezza indicate nella risposta alla domanda (a). Si supponga che la crittografia sia perfetta (ovvero non è soggetta ad attacchi di crittoanalisi).

Il protocollo non è vulnerabile:

- la presenza dell' identificatore del mittente nei messaggi inviati ai passi 1 e 2 rende difficile la realizzazione di *replay attacks*.
- la presenza degli identificatori zero e one nei messaggi inviati ai passi 3 e 4 rende difficile la realizzazione di *reflection attacks*.

[6giugno2006]

> Si consideri il seguente protocollo:
>
> 1. B→A: B
> 2. A→B: $\{Na\}_{Kab}$
> 3. B → A: $\{ f(Na) \}_{Kab}$
>
> dove Kab è una chiave segreta condivisa da A e B, Na è un numero generato con un generatore di numeri pseudo-casuali, e f è una funzione nota sia ad A che a B.
>
> Si assuma che l'agente che esegue il protocollo impersonando il ruolo di A non possa eseguire lo stesso protocollo impersonando il ruolo di B.
>
> Ad esempio, B potrebbe essere il ruolo giocato dal telecomando della vostra macchina, mentre A quello giocato dal sistema di apertura istallato sulla vostra macchina.
>
> Si discuta perchè la sicurezza del protocollo dipende dal numero di bit utilizzati per rappresentare il numero Na.

Se la nonce non è sufficientemente lunga allora la probabilità che Na assuma lo stesso valore in esecuzioni successive del protocollo non è trascurabile e un attacker non deve fare altro che osservare e memorizzare un certo numero di coppie di valori 〈{Na}Kab,{f(Na)}Kab〉 prodotte dal telecomando originale e quindi iniziare il protocollo (facendo finta di essere B, ovvero il telecomando) fino a che A (la macchina) invia un valore {Na}Kab già osservato in precedenza. (Come detto sopra, la probabilità che ciò avvenga è tutt'altro che trascurabile.)

A questo punto l'attacker non deve fare altro che inviare il valore trasmesso in precedenza dal telecomando originale in risposta a quella specifica sequenza di bit.

[cs-esercizi-esame]

> Suppose Alice wants to send her Bank a message that includes her promise to pay Charlie $50 dollars. Alice and the Bank have a shared secret X.
>
> Alice initiates a conversation with the Bank by sending:
>
> A || B || n (Alice's identity, the Bank's identity, and a nonce).
>
> (a) Specify a valid reply for the Bank (i.e., a message generated by the Bank to be sent to Alice) that would enable Alice to verify that the reply came from someone who knows the secret X.

B → A : A || B || n || Hmac(X, A || B || n)

La banca invia ad Alice il messaggio originale più un hash di quel messaggio firmato con il segreto 'X' così Alice può verificare l'hash ed essere sicuro di paralre con la banca.

> (b) In order to setup a secure communication, Alice and the Bank need a secret session key. Suppose that the Bank chooses a session key K by XORing some pseudorandom data with X and includes it with the reply in step (a) above.
>
> Extend message (a) to provide the session key to Alice securely as well.

B → A : A || B || n || E(X,K) || Hmac(X, A || B || n)

Al messaggio precedente aggiungo il valore della nuova chiave di sessione, cifrandolo con il segreto condiviso 'X'.

> (c) Now, Alice can submit her message ("Pay Charlie $50 from my account") to the Bank. Write the message in such a way that Charlie cannot replay it (Hint: you will need to add something to the message that the Bank is capable of checking to prevent replay.).

A → B : A || E(K, M||c) || Hmac(K, A||M||c)

Alice aggiunge un *counter* al messaggio in modo tale da evitare replay attacks.

Access Control exercises

[cs-esercizi-esame]

> Consider the Bell-La Padula security model. Indicate the permissions granted to users with the following security clearances:
>
> 1. (topsecret, {red,blue})
> 2. (secret, {red})
> 3. (secret, {red,black})
> 4. (secret, {red,blue})
> 5. (confidential, {red, blue})
> 6. (confidential, {blue})
> 7. (topsecret, {red,green,blue,black})
>
> over a resource with security label: (secret, {red, blue}).

[Attenzione se è utenti su un file o files ripsetto ad 1 utente. In questo caso: "utenti su un file"]

NoReadUp & NoWriteDown! (obj = oggetto; subj = soggetto)

READ: Clearance_obj è $\leq$ di Clearance_subj $\wedge$ Category_obj è contenuto in ($\subseteq$) Category_subj? Se entrambe vere ha permessi di lettura

WRITE: Clearance_subj è $\leq$ di Clearance_obj $\wedge$ Category_subj è contenuto in ($\subseteq$) Category_obj? Se entrambe vere ha permessi di scrittura

1: solo lettura (user: top-secret; resource: secret quindi solo read. {Red,blue} ok)

2: solo scrittura (perchè: category_subj è contenuta in category_obj)

3: nulla (secret è ok, ma la categoria no.)

4: lettura e scrittura

5: scrittura

6: scrittura

7: lettura