Recap Computer System Security (02KRQOV)

Jacopo Nasi
Computer Engineer
Politecnico di Torino

I Period - 2018/2019

October 29, 2018

Contents

1 Introduction Security ICT System **4**

2 Basic of ICT security **11**

2.1 Cryptography Introduction 11
2.2 Symmetric Cryptography 12
2.3 Asymmetric Cryptography 18
2.3.1 RSA 19
2.3.2 Key Distribution 21
2.3.3 Diffie-Hellman 21
2.4 Elliptic Curve Cryptography 23
2.5 Integrity 24
2.6 Authentication 26
2.6.1 Symmetric Cryptography 26
2.6.2 Asymmetric Cryptography 29
2.6.3 Analysis 30
2.6.4 Public Key Certificate 31

License

This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License.
You are free:

- **to Share**: to copy, distribute and transmit the work
- **to Remix**: to adapt the work

Under the following conditions:

- **Attribution**: you must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work)
- **Noncommercial**: you may not use this work for commercial purposes.
- **Share Alike**: if you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.

More information on the Creative Commons website (http://creativecommons.org).

Acknowledgments

Questo breve riepilogo non ha alcuno scopo se non quello di agevolare lo studio di me stesso, se vi fosse di aiuto siete liberi di usarlo.
Le fonti su cui mi sono basato sono quelle relative al corso offerto (**Computer System Security (02KRQOV)**) dal Politecnico di Torino durante l'anno accademico 2018/2019.
Non mi assumo nessuna responsabilità in merito ad errori o qualsiasi altra cosa. Fatene buon uso!

1 Introduction Security ICT System

Why is securoty an important issue? Nowadays that everything is online and connected to a world wild network, the security over the ICT system has become fundamental. A lack of the secutiry could generate lose for milions of money. Also data breach become a problem.
Everydays technology improve and drive innovation but security must be improved together with the innovations.
With the increase of the number of connected devices, the IoT (Internet of Things), security start to facing a lot of more problem, the complexity of the scenario has become really really big. From personal devices, like desktop, laptop, fridge or car, by communications networks, and to distributed services, everything must be secured!

Complexity enemy of security based on one of the first axiom of engineering: *"The more complex a syste, is, the more difficult its correctness verification will be."*. Keep a system as simple as possible is always a good idea. The KISS rules (***Keep It Simple, Stupid***) is one of the most important rule over the system security.

Definition of ICT Security "It is the set of products, services, organization rules and individual behaviours that protect the ICT system of a company.

It has the duty to protect the resources from undesired access, guarantee the privacy of information, ensure the service operation and availability in case of unpredictable events (C.I.A. = Confidentiality, Integrity, Availability).

The objective is to guard the information with the same professionalism and attention as for the jewels and deposit certificates stored in a bank caveau.

The ICT system is the safe of our most valuable information; ICT security is the equivalent of the locks, combinations and keys required to protect it."

— **Italian Bank**

An important part of the security study is the Risk Estimation, is a fundamental step that take in account all the assets and events to evaluate the risk of something. The flow is showed in figure 1:

Where the assets is composed by everythings needed by a service to work, both soft and hard part, also human resources. The vulnerabilities, intrinsic of an asset, represent the weakness of it. The threats is a deliberate action, or an accidental event, that can produce the loss of a security properties exploting a vulnerability. The event is also characterized by an impact and a probability

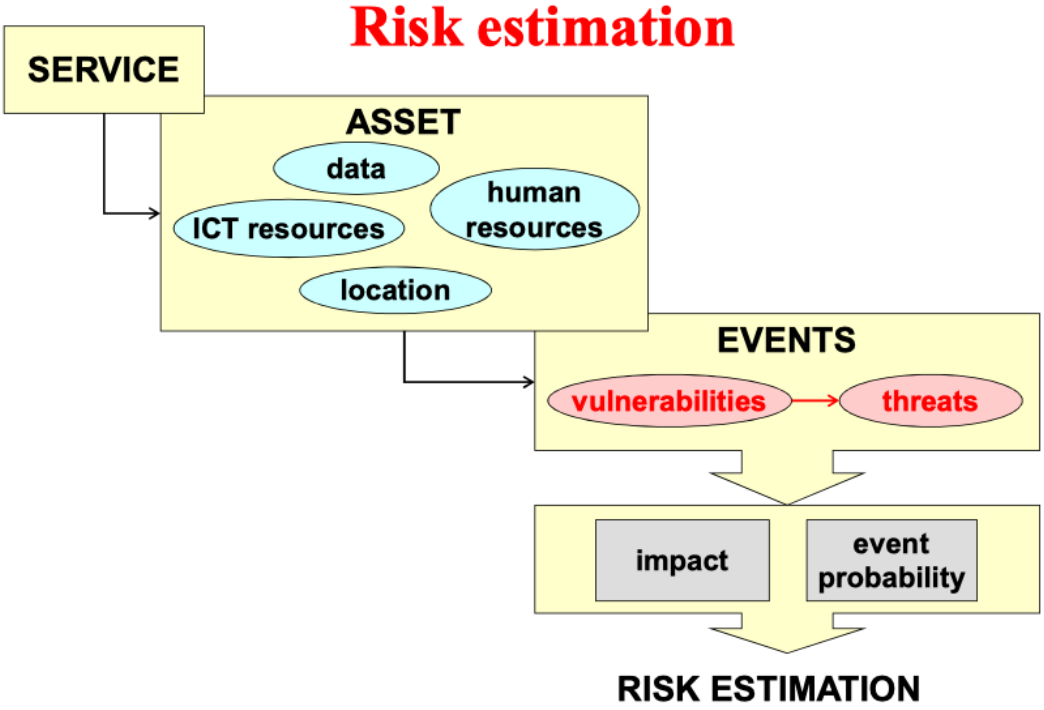

Figure 1: Risk Estimation

that could be high, low or other middle values.
Direct following of the risk estimation is the Analysis and management of security. After the evaluation of risks, is necessary to:

1. Select Countermeasures
2. Implement Countermeasures
3. Audit (check if works)

The security implement is not a phase of the developmente process, is part of each sigle part of it. Security can't be compute at the end of the development, it must be implement from the beginning of the process. **Security is a process, not a product!** The following figure 2 show the parallel line followed by the security development.

An important definition, before speaking about security itself, is the **Window of Exposure** the represent the time when an attack could be perfomed and there are no countermeasures to avoid it. This window could potentially be infinite and this is the real problem. The figure 3 show how this windows id divided in different part:

As already says, security is not a product but is a proccess. Computer flaws are inevitable and this is way we can't use devote our security to only secured products. The only way to effectively do business is an insecure world is to put processes in place that recognize the inheritent insecurity in the products. **The**

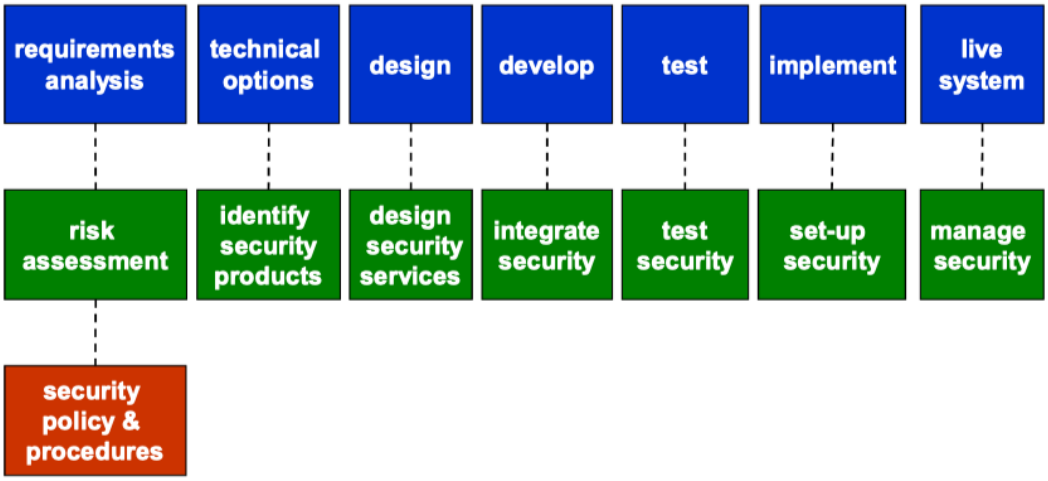

Figure 2: Security Life Cycle

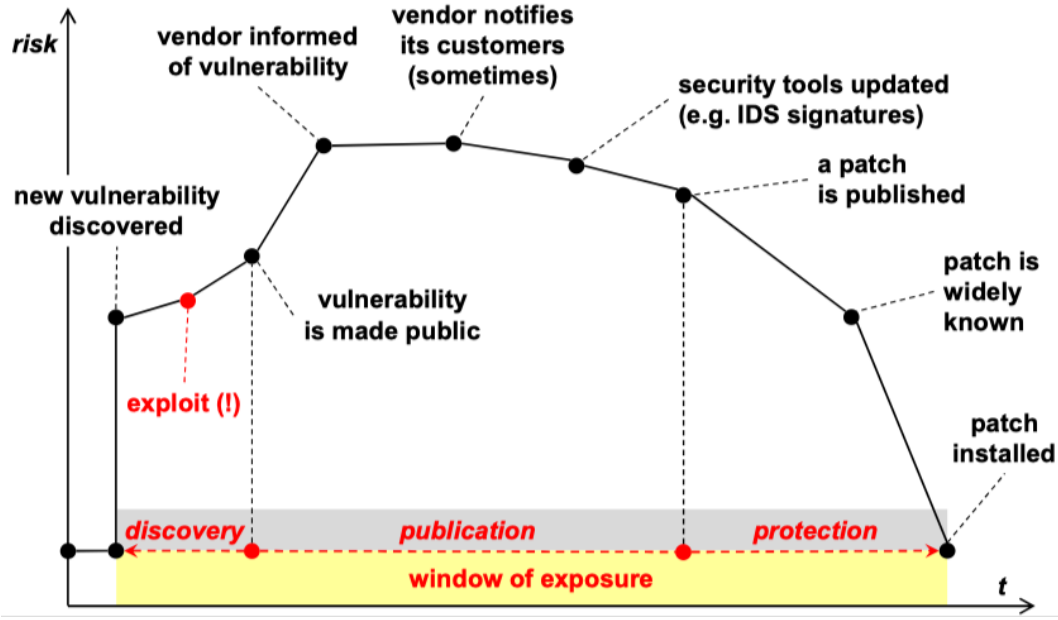

Figure 3: Window of Exposure

trick is to reduce your risk of exposure regardless of the products or patches.

Security Principles Here some of the most important security principles:

- Security by Design
- Least Privilege: Only correct rights and the only few needed
- Need-to-know: Access to the only piece of stuff that are needed

- Security by Default
- Security in Depth: More the importance of the system, more the obstacles

Security Properties The following list show the most importante properties of the security world:

- Authentication (Simple/Mutual): Source (or both) must prove themself
- Data Origin/Authentication
- Authorization and Access Control
- Confidentiality/Privacy/Secrecy
- Non Repudiation: Formal proof, acceptable by a court of justice, that gives undeniable evidence of the data creator
- Availability
- Accountability
- Integrity: Modification, Filtering

Where is the enemy? The enemy normally is supposed to be outside of our system but is not simple as it seems. The possible locations are:

- Outside our organization (Firewall)
- Outside our organization, with exceptionn of our parters (VPN)
- Inside our organitazion
- Everywhere!

The last item is probably the more true. The distinction between internal/external and good/bad guys is no more sufficent. From the *Verizon Data Breach Invetigation Report* the percentage of source of attacks is: 20% Internal and 80% External, probably the internal percentage is a little bit higher due to the fact that Verizon is a provider and can't show the internal side so much.

Basic Problems There are some basic problem for security:

- Networks are insecure:
  - Clear communications
  - LAN use broadcast
  - Not E2E geographical connections

- Weak user Authentication
- No server Authentication
- **Software with bugs!**

Classes of Attacks Some of the most common type of attacks:

- IP Spoofing / Shadow Server
- Packet Sniffing
- Connection Hijacking / Data Spoofing
- Denial-of-Service (Distributed DoS)

The **IP Spoofig**, or source address spoofing, is forginng the sourcce network address, typically performed at LV3 (IP), but also at LV2 can be performed. The typical attacks are: Data Forging and unauthorized access to systems.
The **Packet Spoofing** it reads the packet addresses to another network node, it easy to do in LAN or at the switching nodes. It allows to intercept password, data and other stuffs.
The **Denial-of-Service (DoS)** it keeps a host busy so that it can't provide its services. There are a lots of examples: mail/log saturation, ping flooding, SYN attacks. The main problem of this kind of attacks is that there are no countermeasures. The **Distributed DoS** are similar to the previous one but perfomed by a greater number of hosts (botnet), controlled by a master. The power is the same of a normal DoS multiplied by the number of deamons of the botnet. There are also some techniques to improve the attack like using a reflector to hide the attacker's track. One of the more important DDoS was performed agains Yahoo! during Feb 2000.
The **Shadow Server** is a technique that host that manage the attacks show itself (to victims) as a service provider without having the right to do so. It provide a "wrong" serice to victims, like bank sites or other stuffs.
Connection Hijacking / MITM, AKA Data Spoofing, is performed when attacker takes control of a communicatio channel to insert, delete, or manipulate traffics. It can edit data is all forms, and change the messages of the communications. Another similar for is the **Trojan / MITB**, it used the fact that network channels are more protected, but users terminals not. The behaviour is to install a keylogger to store everything. It could be also passed via browser extension.
The are other application-leve problems:

- Buffer Overflow
- Cookies
- Clear password in DB

- ”invent” a protection system

We make now some clearance about name of malwares:

- Virus: Damage the target and replicate itself, propagated by humans (require complicity)
- Worm: Damage target because replications (resource saturation)
- Trojan: Malware vector
- Backdoor: Unauthorized access point
- Rootkit: Privileged access tools, hidden and stealth
- Ransomware: Make hosts unreadable (can be also silent)

Non technological problems Prorably the greatest majority of the problems and leaks come from here! The are some basic problems: due to low awareness, mistakes, tendency to trust and other facts the major cause of leaks are humans.
The **Social Engineering** are sets of techniques used to asks the involutary user's partecipatio to the attack action, usually the naïve users are targeted (e.g. *“do change immediately your password with the following one, because your PC is under attack”*), also experienced users are targeted (e.g. by copying an authentic mail but changing its attachment or URL).
One of the most used technique is the **Phishing** (fishing) where the attacker try to stole information (fishing) from the target (fish), it can be achieved, for example, by showing acquaintance with the company's procedures, habits and personnel helps in gaining trust and make the target lower his defences. Is often performed by using fake mail, SMS or IM. The normal procedures works by attarcting the fish in a shadow server where it will leave of the sensitive informations or persuade to install plugins or other stuffs. Two variants exist, **spear phishing** (include several personal data to disguise the fake message as a good one, e.g. mail address, name of Dept/Office, phone no.) or **whailing** (targeted to VIP such as CEO or CIO). The **Pharming** is a set of several techniques to re-direct a user towards a shadow server, chaging the hosts file, nameserver, poisoning cache of DNS. Some important examples of this techniques are T.J.Maxx attack, Transformed3 phishing or Stuxnet.
The typical path of attackers is showed in figure 4.

From this brief introduction we can define he three pillars of security:

Figure 4: Cyber (intrusion) Kill Chain

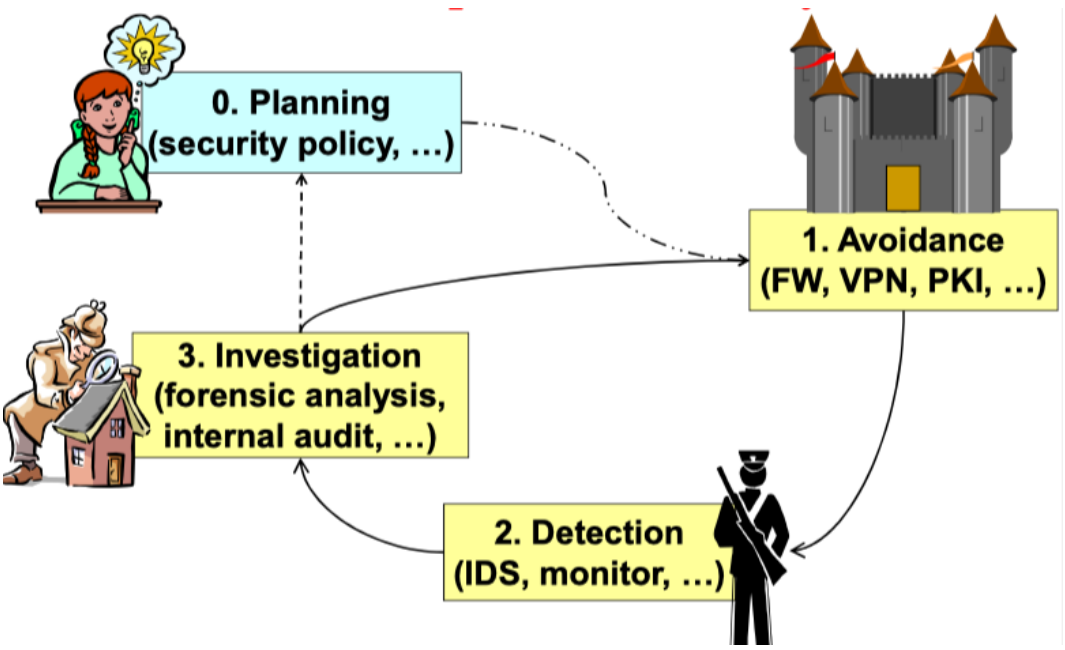

Figure 5: Pillars of Security

and we can define the main figures:

- Hacker: Good and skilled
- Cracker: Bad but skilled

- Script Kiddie: Bad but NOT skilled
- Wannabe Lamer: Not good at all

2 Basic of ICT security

2.1 Cryptography Introduction

The basic principles of Cryptography are showed in the following figure

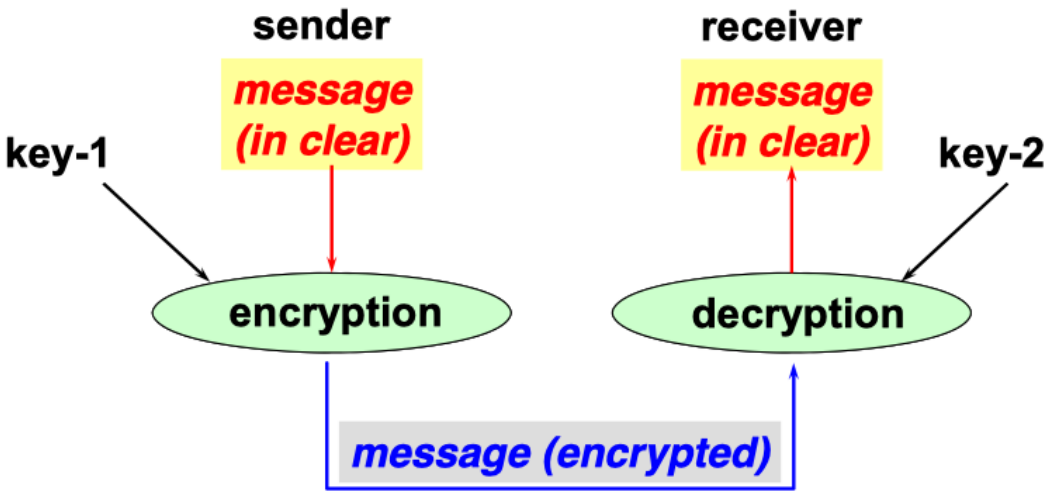

Figure 6: Cryptography flow

Some important terminology. The message in clear is called:

- Cleartext or plaintext
- Refer with **P**

Instead the encrypted message:

- Ciphertext
- Refer with **C**

Some principles of Cryptography, written by Kerchoffs, are: If the keys:

- Are kept secret
- Are managed only by trusted systems
- Are of adeguate lenght

then...

- it has no importance that the ecryption and decryption algorithms are kept secret
- on the contrary it is better to make the algorithms public so that they can be widely analysed and their possible weaknesses identified

In computer system STO (Security Through Obscurity) is not good.
An important operator of this world is the XOR function that is the ideal confusion operator, because it not change the probability. Is also a primitive operation present in all CPUs.

2.2 Symmetric Cryptography

Are all the algorithms based on a secret key shared between sender and receiver, used for encrypt and decrypt. Figure 7 show the general flow of this kind of algorithms. The advantage are low computational cost, in fact is used for data encryption.

- $C = enc(K, P)$
- $P = dec(K, C) = enc^{-1}(K, C)$

One of the main problem is *"How to share (securely) the secret key among sender and receiver?"*. T

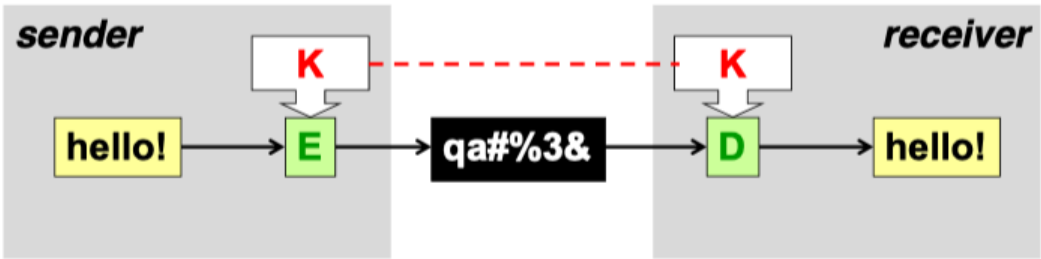

Figure 7: Symmetric General flow

There is not a confirmation of correct decryption, is up to the user to understand if the result is correct or not, in any case a result is provided with any key.

DES (Data Encryption Standard) one of the first used by US government. Is now obsolete, is based on a key of 64 bits, composed by:

- Key: 56 bits
- Parity: 8 bits

This means that the actual bit of resistence is 56. It's based on 64 bits of data blocks. It is also designed to be efficent in hardware with poor performance, the flow is:

1. XOR
2. Shift
3. Permutation (not so good performance)

3DES is the triple repeted application of DES, it used two of three 56 bits keys. Is different from passe to a keys long 56*3 because the key lenght will not match with that number of bits. It can be computed in two ways:

- 2 Keys: $C = enc(K_1, dec(K_2, enc(K_1, P)))$
- 3 Keys: $C = enc(K_3, dec(K_2, enc(K_1, P)))$

The central step of decryption is made to generated mess.
An important issue of the encryption is the doubling of an algorithms. Double application of encryption algorithms is subject to a know-plaintext attack named **meet-in-the-middle** which allows to decrypt data with at most $2^N + 1$ attempts (if the keys are N-bits long). Thus the double version is never used, it double the computation time and increase they key lenght of only one bit. The formulas are:

$$C = enc(K2, enc(K1, P))$$

$$dec(C, K2) = dec(K2, enc(K2, enc(K1, P)))$$

$$dec(C, K2) = enc(K1, P)$$

The attacker can compute ENC(K1, P) for all values of K1 and DEC(K2, C) for all possible values of K2, and add only 1 bit of strenght for 2 keys of the same size.

IDEA International Data Encryption Algorithm it was patented but with low royalty, developed for 16 bits architectures. It use a 128 bits key and 64 bits of data block, it's famous because is used in PGP. The operation used are:

- XOR
- Addition Modulo 16
- Multiplication Modulo $2^{16} + 1$

RC2, RC4 other algorithms developed by Ron Rivest (Ron's Code), they are algorithm proprietary of RSA but not patented, 3 to 10 times faster than DES. RC2 is a block algorithm, RC4 is a stream one. They are using a variable length key.

Application of block algorithms applying these algorithm over data of size different from the block size require a little effort. When the data size is greater that the algorithm's block size:

- ECB (Electronic Code Block)
- CBC (Cipher Block Chaining)

In the other case:

- Padding
- CFB (Chiper FeedBack), OFB (Output FeedBack)
- CTR (Counter Mode)

The frist solution **ECB** is a bad idea, it based on the division of the data in chunks with the same size of a block, by encrypting them with a K (key), the formula is: $C_i = enc(K, P_i)$. The problems of this solution are:

- Swapping of two blocks of cipher goes undetected
- Identical blocks generated identical cipher texts hence it is vulnerable to *know-plaintext* attacks. (Word example)

These are the main reason to avoid the use of this solution for big data. The decryption is made by reversing the process. The figure 8 show the process.

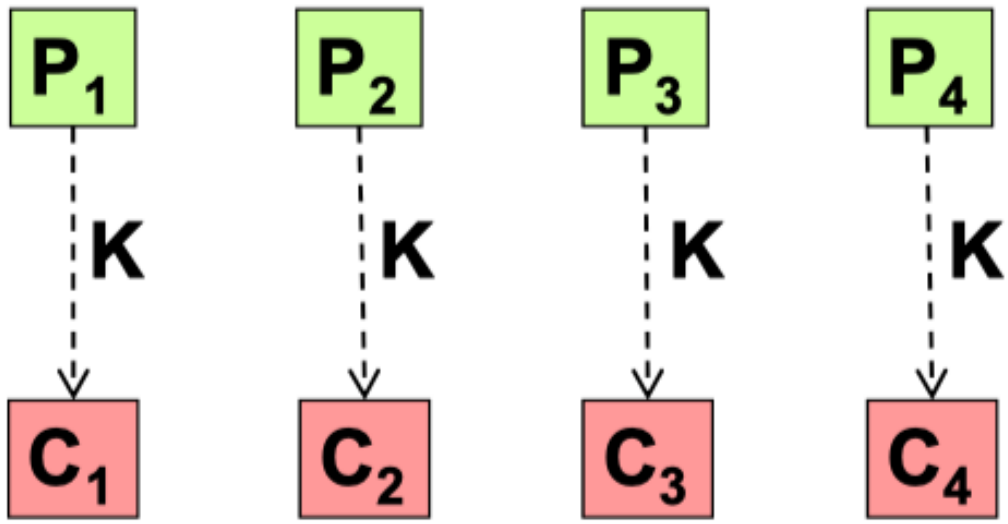

Figure 8: ECB

The solution of **CBC** [figure 9] is more secure, by using an IV (Initialization Vector), it encrypt the block by XORing it with the previous block and with

the key, this avoid the problem of the *know-plainntext*, because same text in different position will generate different encrypted text. The IV is added for increasing the difficulties of decryption. The formula is the following:

$$C_i = enc(K, P_i \oplus C_{i-1})$$

The decryption require the Initialization Vector (C_0), the formula is the revert the process by the following formula:

$$P_i = dec(K, C_i) \oplus C_{i-1}$$

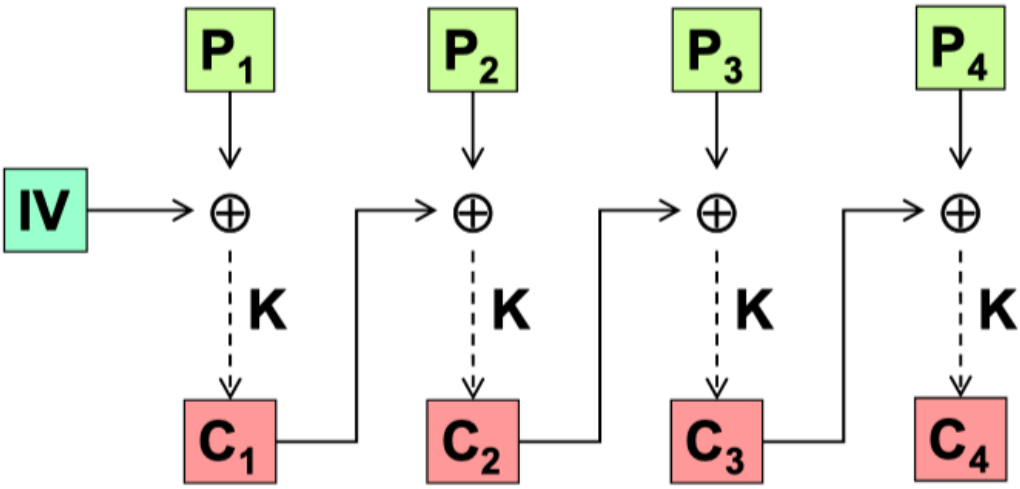

Figure 9: CBC Encryption

When the data is shorted than the block size, a **padding** (or aligning, filling) is required. It consist by adding some data at the end to completely fill the block. The are several techniques used for padding:

- If the length is know: 0x00 bytes
- Original DES: 1 bit followed by 0
- One byte 128 (0x80) followed by null 0x00
- Last byte value equal to length of padding

The are also some techniques with explicit length for padding:

- (SSL/TLS) bytes with value L
- (SSH2) random bytes
- (IPsec/ESP) progressive numbers

Padding is typically applied to large data, on the last fragment resulting from the division in blocks (ECB or CBC), for $|D| < |B|$ we prefer ad hoc techniques like CFB, OFB or CTR. Another important note is that, even if the plaintext is an exact multiple of the block, padding must be added anyhow to avoid errors in the interpretation of the last block.

CTS (CipherText Stealing) permits to use block algorithms without padding:

- Last (partial) block filled with bytes from the second-to-last block
- These bytes are removed from the second-to-last block (which become partial)
- After encryption, exchange the position of the last and second-to-last blocks.

The figure shows better the solution algorithm:

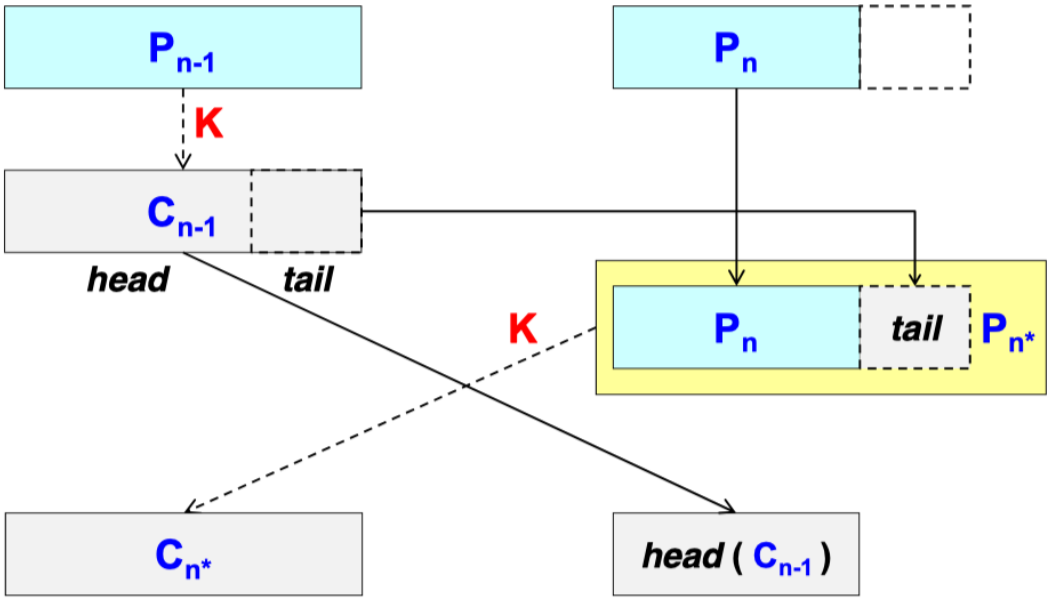

Figure 10: CTS with ECB Encryption

This technique is useful when we cannot increase the size of the data after the encryption. The tail, store, after the encryption in the second-to-last block is encrypted 2 times and it requires 2 decryption.
The **CRT** (Counter mode), really used, uses a block algorithm to cipher N bits at a time. It require:

- **NONCE** = Number used ONCE
- counter

The flow is showed in the following figure 11:

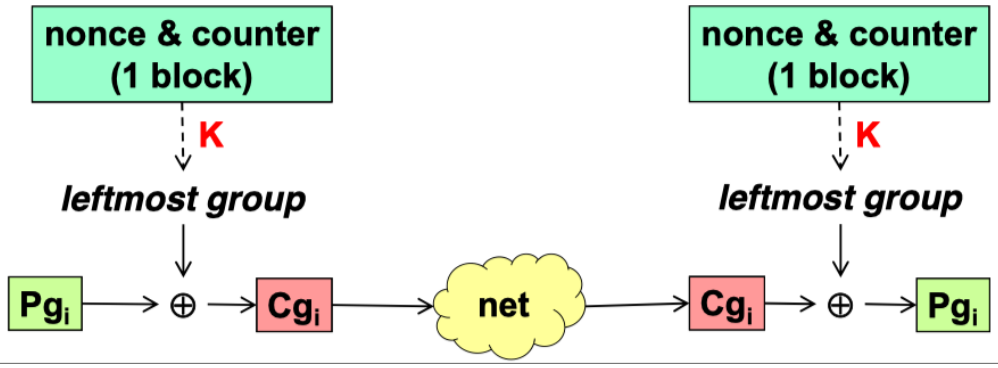

Figure 11: CRT flow

Symmetric Stream Algorithms this kind of algorithms not require to separate datas in blocks. They operate over one bit/byte at the time. The ideal algorithms require a key which is as long as the message to protect, of course this is not possible. The real algorithms use pseudo-random key generators, synchronized between the sender and receiver, some examples are RC4 and SEAL. The idea is show in figure

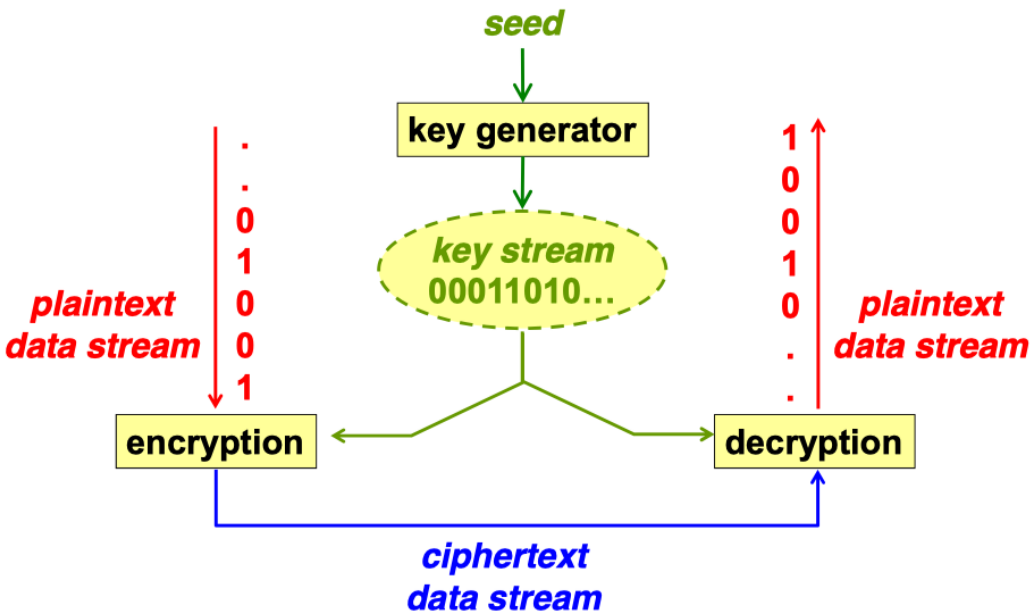

Figure 12: Algorithms of type stream

A lot of challenges were started for stream ciphers, some algorithms are:

- HC-128, Salsa, SOSEMANUK
- Grain v1, MICKEY 2.0

- Salsa20 and ChaCha20
- Camellia
- SEED and ARIA

Problem of symmetric cryptography one of the main problem of all this technique is that one key is required for each couple / group of user in order to guarantee privacy within groups. This means that, for a complete private communication between N parties, $N * (N - 1)/2$ keys are necessary, these require key exchange algorithms and generate a problem for large groups.
Another know problem is the **length of secret keys**, if:

- The encryption algorithm was well designed
- the keys - N bit length - are kept secret

... then the only possible attack is the brute force (exhaustive) attack which requires a number of trials equal to:

$$T = 2^{Nbit}$$

This means that and increase of the computational power, reduce the strenght of the keys, that require a growning number over years. For this reason a challenges (AES) was promoted to develop another algorithm, the winner was **Rijndael**.

2.3 Asymmetric Cryptography

The Asymmetric Cryptography is based on 2 different keys generated in pairs, the two keys are different, one is called *public* (**Kpub**) and the other *private* (**Kpri**). If one key is used for encryption then the other one must be used for decryption, they have an inverse functionality. The computation load is high, due to this fact asymmetric encryption is used only for data streams and should not be used for data storage. The figure 13 show the process. The principals algorithms are:

- Diffie-Hellman, RSA, DSA, El Gamal, etc...

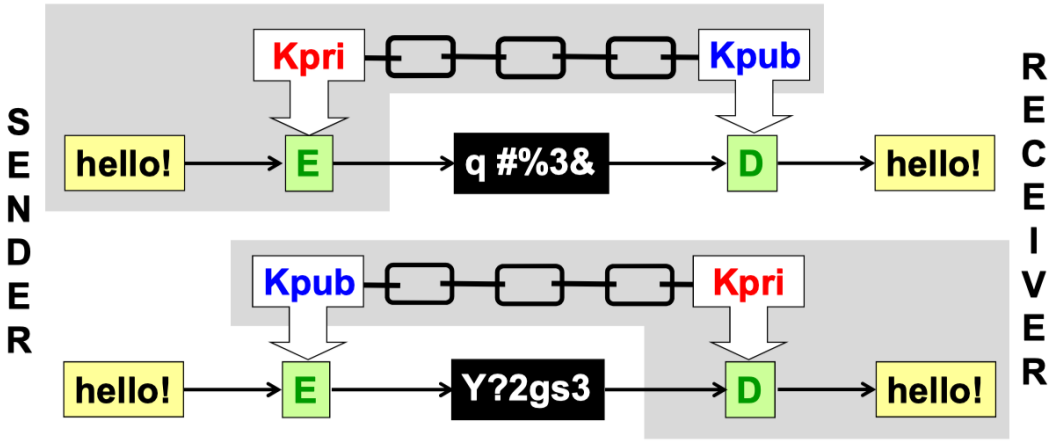

Figure 13: Asymmetric Cryptography with pairs of keys

Another possible application os this kind of cryptography is the **Digital Signature**, used to provide data authentication and integrity. Usually not all the data is encrypted but only its summary (***digest***).
Another important feature is the possibility to make secret message for a particular receiver given only its public key.

2.3.1 RSA

This algoritmh is based on a public module:

$$N = P * Q$$

known to anybody P and Q are ***prime number, large and secret***. Another value, $PHI = (P-1)(Q-1)$, along a public exponent E such that arbitrarily $1 < E < PHI$ and it is relative prime with respect to PHI.
From these element, a private exponent is computed:

$$D = E^{-1} mod(PHI)$$

from this last value the 2 keys can be computed:

- public key = (N, E)
- private key = (N, D)

After that computation P and Q must be deleted, discarded, killed!!!
Otherwise the process can be inverted.
An important note about RSA is that it can cipher/decipher only data whose value is less than the value of the module N, similar to a block algorithm but with a variable lenght. The final result is computed:

- encrypt: $c = p^E mod(N)$

- decrypt: $c = p^E mod(N)$

For matematichal reasons the roles of the 2 exponents, E and D, are interchangeable:

$$(x^D)^E mod(N) == (x^E)^D mod(N)$$

The advantage of using modular arithmetic is that the inverse of a reminder of a module could be "any number", there are infinite possibilities. An example in figure 14:

chosen P=11, Q=19 we have N=209 and PHI=180

E (relative prime to 10 and 18 and <180) = 31

$D = 31^{-1}$ mod 180 = 151

Kpub = (209, 31) ; Kpri = (209, 151)

text to encrypt: 10 13 23 (note: p_i < 209)

$c_1 = 10^{31}$ mod 209 = 32

$c_2 = 13^{31}$ mod 209 = 167

$c_3 = 23^{31}$ mod 209 = 199

$p_1 = 32^{151}$ mod 209 = 10

$p_2 = 167^{151}$ mod 209 = 13

$p_3 = 199^{151}$ mod 209 = 23

Figure 14: RSA an example

During years some optimizations were developed, like the one based on the CRT (Chinese Remainder Theorem) that makes the procedure 4x times faster.

Weaknesses There are some know weaknesses:

- Small encryption exponent
- Same keys for encryption and signing
- Acting as an oracle

Nowadays the lenght of the keys is important:

- 512 bits keys $\sim$ Some weeks

- 1024 bits keys ∼ Some Months
- **2048 bits keys ∼ Several Years**

During Eurocrypt of 1999 a devices for decrypting RSA faster was annouced, it has never been presented, probably for interest reasons. For that, some governments, no longer want to use RSA solutions.

2.3.2 Key Distribution

The distribution is a fundamental step of asymmetric cryptography, the private key must kept SECRET, the ublic instead must be distributed as widely as possible. But how is possible to guarantee the binding between public key and person identity? There are 2 possible solutions:

- Exchange of keys Out-Of-Band (e.g. keys party)
- Distribution by specific data structure (public key certificate)

Confidentiality without shared secrets is often used to send the secret key chosen for a symmetric algorithm, looking at figure 15.

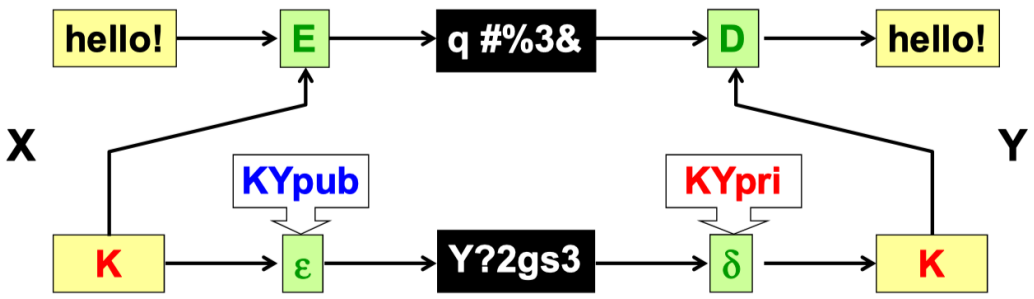

Figure 15: Confidentiality, no shared secret

2.3.3 Diffie-Hellman

Is the other well-know algorithm for asymmetric encryption. The idea behind the DH algorithm is good represented by figure 16.

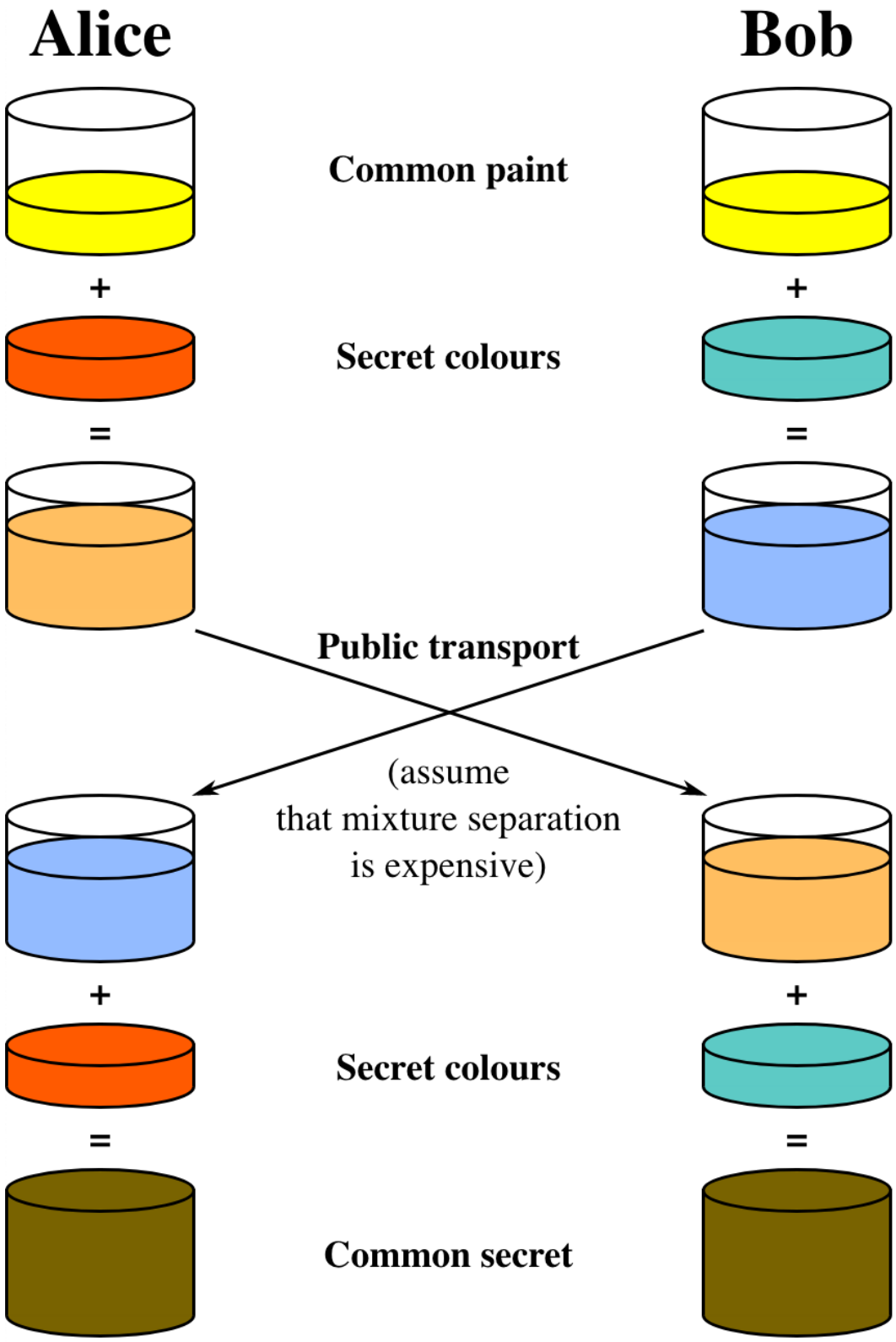

Figure 16: Diffie-Hellman idea

It was the first developed public-key algorithm, is frequently used to agree on a sceret key (*key agreement*), it was patented but is now expired. One advantage is that is sniffing resistant. It can be exploited by a MITM attack by manipulating the data like in figure 17.

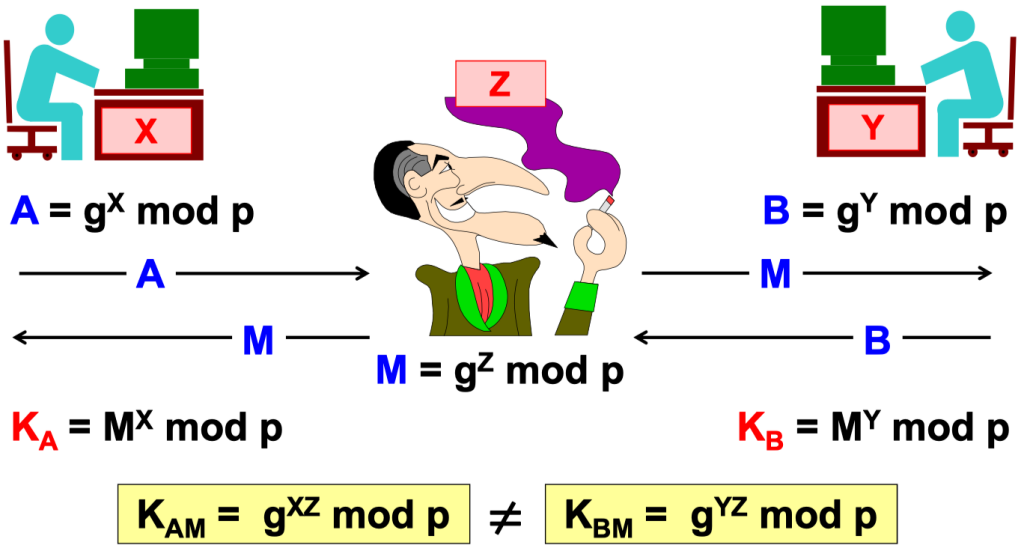

Figure 17: Diffie-Hellman MITM

The complexity of DH is a discrete logarithm, in any case also RSA, required the usage of quantum computer to be exploited in a reasonable amount of time.

2.4 Elliptic Curve Cryptography

This solution instead of using modular arithmetic, the operations are executed on the surface of a 2D (elliptic) curve. The problem of discrete logarithm on such a curve is more **more complex** than modular and allows **shorter keys** (about 1/10). A lot of algorithms has been rewritten, **EC**DSA or **EC**DH.

ECDH the EC version of Diffie-Hellman is the same as before but with the math of an elliptic curve.

- A and B select same elliptic curve and a point G of its
- A chose a random value x and computes: $X = xG$
- B chose a random value y and computes: $Y = yG$
- A and B exchange (publish) X and Y

- A computes $K = xY$
- B computes $K' = yX$

but the final result will be that:

$$K = K' = xyG$$

There are also other algorithms like ECDSA:

- Message digest computed with normal hash function (SHA-256)
- Signature = pair of scalars derived from the digest plus some operations on the curve

The ECIES instead:

- Generates a symmetric encryption key (AES-128) with operations on the curve
- Gives to the receiver the information (based on his public key) needed to recompute the encryption key

2.5 Integrity

In most of the cases the real needes of the people around the world is not the secrecy of the data, but is its integrity. Intercepting a communication and change it, in an unpredictable way, could be worse than read it. For this reasons the digest come in.
The **digest** is a summary of something, the name is related to the *reader digest* a magazine with the *summary* of famous books.
The message digest is a fixed-length summary of the message to be protected (of any lenght). Is important for a digest to be of a fixed-size without depending upon the lenght of the data to be summarizzed. It must be:

- **FIXED-SIZE**
- Fast to compute
- Impossible (or very difficult) to invert
- Difficult to create "collisions"

The digest is often used to avoid performing operations on the whole message, especially when the message is very large (slow pub-key crypto). The digest can be calculated in many ways, usually is calculated via ***(Cryptographic)*** **Hash Function**. The hashing algorithms are the faster in the world of cryptography, more than the symmetric ones and more than asymmetric, that are the slower.

The flow is simple (figure 18), the message M is splitted in N blocks $M_1 ... M_N$, the blocks are then computed by a base function f. Each block generate and hash code:

$$V_k = f(V_{k-1}, M_k)$$

For the first block the $V_0 = IV$ (Initialization Vector), public and specified by the algorithm code. The digest is generated by the last computation of the function:

$$hash = V_N$$

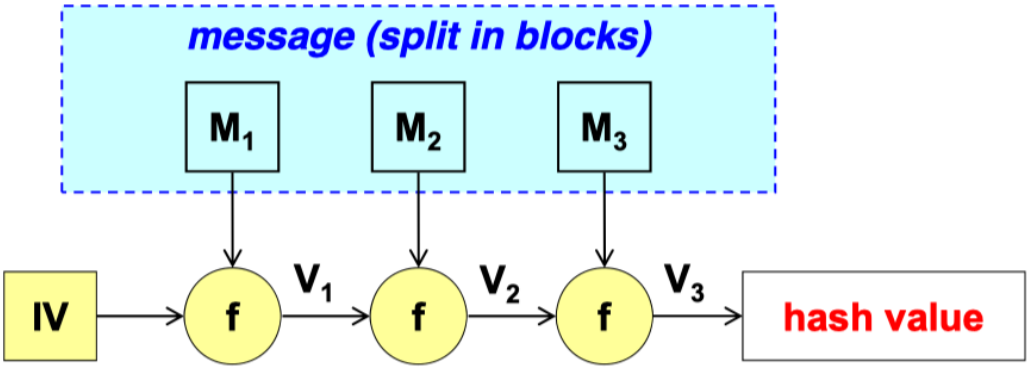

Figure 18: Hashing flow

The table show some advice about digest algorithms:

name	*block*	*digest*	*definition*	*notes*
MD2	8 bit	128 bit	RFC-1319	obsolete
MD4	512 bit	128 bit	RFC-1320	obsolete
MD5	512 bit	128 bit	RFC-1321	obsolete
RIPEMD-160	512 bit	160 bit	ISO/IEC 10118-3	good
SHA-1	512 bit	160 bit	FIPS 180-1	sufficient
SHA-224	512 bit	224 bit		
SHA-256	512 bit	256 bit	FIPS 180-2	good
SHA-384	512 bit	384 bit	FIPS 180-3	
SHA-512	512 bit	512 bit		
	SHA-2			
SHA-3	1152-576	224-512	FIPS 202 FIPS 180-4	excellent

Figure 19: Digest Algorithms

The lenght of the digest is important, must be long enough to avoid ***aliasing*** = **Collisions**:

- md1 = h(m1)

- md2 = h(m2)
- if (**m1** $\neq$ **m2**) $\Rightarrow$ md1 $\neq$ md2

For an algorithm well designed the probability of aliasing is:

$$P_A \propto 1/2^{Nbit}$$

for this reasons digest function requires a good number of bits (birthday paradox).

The are some code used to guarantee the integrity of the messages: **MIC** *(Message Integrity Code)*, to guarantee the authentication: **MAC** *(Message Authentication Code)* and to avoid the replay attacks **MID** *(Message IDentifier)*.

2.6 Authentication

The authentication is the process or action of verifying the identity of a user or process, or showing something to be true, genuine, or valid.

2.6.1 Symmetric Cryptography

Authentication can be performed in different ways. One possibility is the **auth by Symmetric Encryption**. This solution send also an encrypted copy of data:

- (sender) M* = enc(K, M)
- (transmission) M || M*
- (receiver) X = dec(K, M*)
- (verification) if (X == M) $\Rightarrow$ OK, else ALARM

Using this technique only who knows the (secret) key can compare the copy with the original. The disadvantage is that the performance are really poor, every operation must be executed 2 times, also the transmission.
Another similar technique is the **authentication by digest and symmetric encryption**, the adavantages are the data transmission is not doubled and only a digest is sent. The drawback is that two operations are required (digest + encryption).

- (sender) H = enc(K, hash(M))
- (transmission) M || H
- (receiver) X = dec(K, H)

- (verification) if (X == hash(M)) ⇒ OK, else ALARM

Is also possible to enforce authentication by means of **keyed-digest**, in this solution the digest sended along with the data is calculated over data an a private key together.

- (sender) d = digest(K, M)
- (transmission) M || d
- (receiver) d* = dec(K, M)
- (verification) if (d == d*) ⇒ OK, else ALARM

The solution require less overhead and only a single operation. One of the most used solution is the **HMAC** where:

$$hmac = H(K' \oplus opad || H(K' \oplus ipad || data))$$

In case of size problem CBC-MAC can be used, it exploits a block-oriented symmetric encryption algorithm, in CBC mode with null IV, taking MAC as last encrypted block. Both these solution are good, the only problem is the secret **shared** key, because the identity of the signer can't be distinguished, for the reasons is not good to be use in a commercial environment.
How is possible to combine both integrity and secrecy?

- Secrecy = Symmetric Encryption with K_1
- Integrity = Keyed-digest (MAC) with K_2

Some possibilities are:

- Auth&Encr (eg. SSH)
  - enc(p, K_1) || mac(p, K_2)
  - Need decrypt before check integrity
- Auth-then-Encr (eg. TLS and SSL)
  - enc(p || mac(p, K_2), K_1)
- Encr-then-Auth (eg. IPsec)
  - enc(p, K_1) || mac(enc(p, K_1), K_2)

Improper combination of secure algorithms may leed to an insecure result!
The AE tecniques allows to achieve both privacy and authentication (and integrity):

- One key

- One algorithm
- Better Speed
- Less error in combining

Are often used in applications, e.g. emails, networks, etc...
The RFC 5116 release define Interface and algorithms for authenticated encryption , for example the AEAD in figure 20.

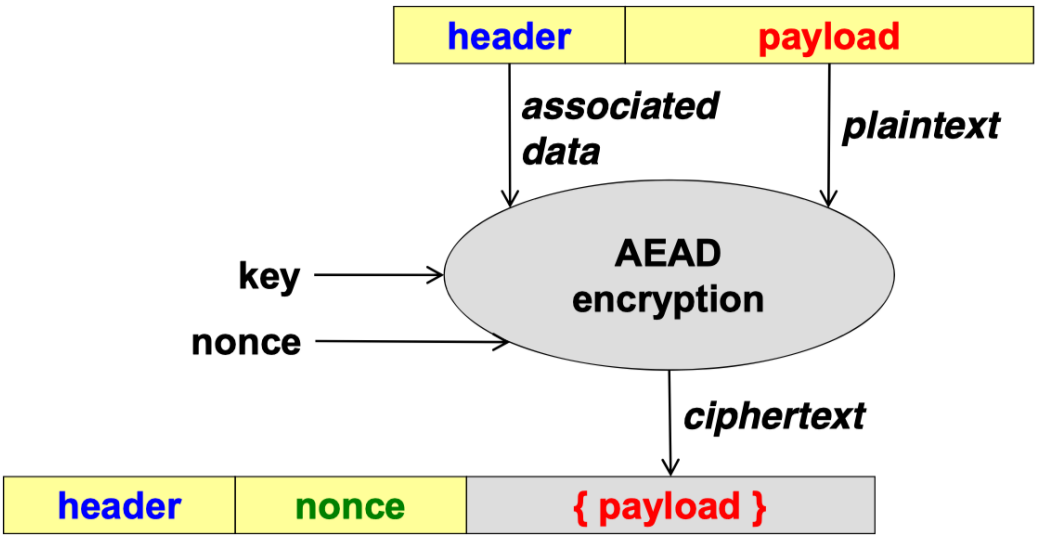

Figure 20: Authenticated Encryption with Asociated Data (AEAD)

The idea of AEAD is that comparing the two header, the one ciphered and the plain one, will guarantee the integrity.
The **IGE** (*Infinite Garble Extension*) is a modde of application and is not too much used because is too simple to be exploited, the flow is in figure 21.

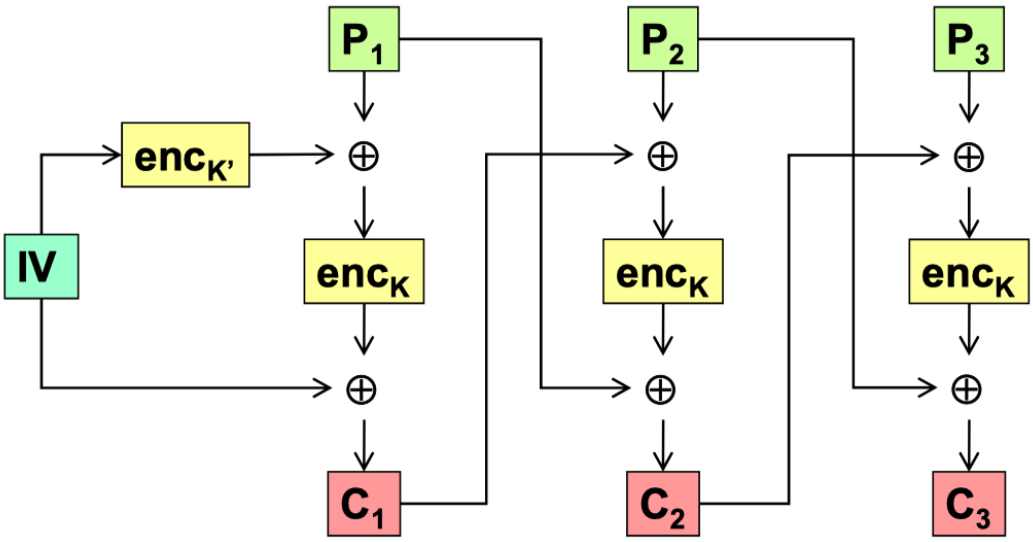

Figure 21: IGE (Infinite Garble Extension)

The AE standard are:

- OCB 2.0
- AESKW (AE Key Wrap)
- CCM (CTR mode with CBC-MAC)
- EAX
- Encrypt-then-MAC
- GCM

2.6.2 Asymmetric Cryptography

Authentication can be also achieved by using **asymmetric** encryption. For example the **auth by digest and asymmetric crypto** is one of the first implementations:

- (signer S) H = enc(S_Kpri, hash(M))
- (transmission) M || H
- (verifier V) X = dec(S_Kpub, H)
- (verification) if(x == hash(M)) ⇒ OK, else ALARM

those who knows the public key of the sender can compare the transmitted digest with the digest calculated on the received data. This is also know as **Digital Signature**! The figure show the behaviour.

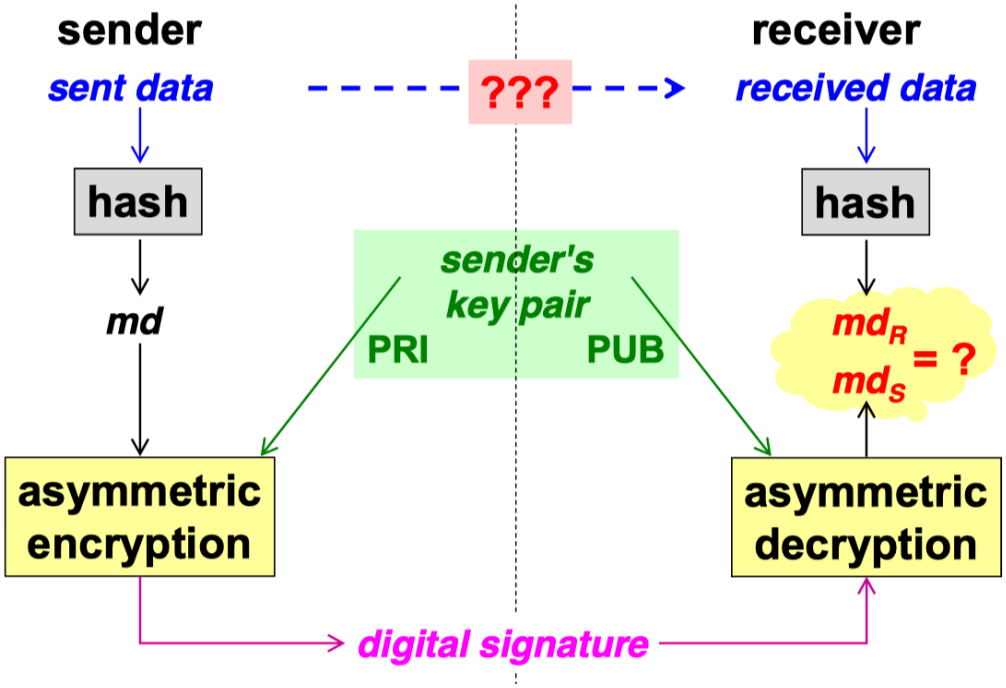

Figure 22: Digital Signature

DS can be achieved with RSA-based algorithms, they must be:

- Resistant to collisions (even just to avoid generating accidentally the same signature)
- Difficult to invert

2.6.3 Analysis

The A-I by means of a shared secret:

- Useful only for the receiver
- Cannot be used as a proof without disclosinng the secret key
- Not useful for non repudiation

Instead, by means of asymmetric encryption:

- Being slow it is applied to the digest only
- Can be used as a formal proof
- Can be used for non repudiation
- = digital signature

The main advantage of a digital signature over an handwritten one is that it can guarantee both authentication and integrity because is tightly bound to the data.

2.6.4 Public Key Certificate

A data structure used to securely bind a public key to some attributes, the idea is to bind a key to an identity, also other association are possible. The key will be digitally signed by the issuer: the **Certification Authority** for a limited amount of time. It can be revoked on request both by the user and the issuer. Some of the most common formats for pub key certificates are:

- X.509: ISO + IETF
- non X.509: PGP
- PKCS#6: RSA

X.509 is oe of the most used certificate format, the structure il composed of:

- Version
- Serial number
- Signature Algorithm
- Issuer
- Validity
- SUBJECT
- SUBJECT PubKeyInfo
- CA Digital Signature

An example in figure 23:

2

1231

RSA with MD5, 1024

C=IT, O=Polito, OU=CA

1/1/97 - 31/12/97

C=IT, O=Polito,
CN=Antonio Lioy
Email=lioy@polito.it

RSA, 1024, xx...x

yy...y

Figure 23: X.509 Certificate Example

The **PKI** (*Public-Key Infrastructure*) is the Infrastructure, technical and administrative, put in place for the creation, distribution and revocation of public key certificates. Any certificate can be revoked before its expiration date:

- By the owner
- By the creator

when a signature is verified, the receiver must (and is a responbilitity) check

that the certificate was valid at signature time. There are 2 type or revocation mechanisms:

- CRL (Certificate Revocation List):
  - List of revoked certificates
  - Signed by the CA or by delegated party
- OCSP (On-line Certificate Status Protocol)
  - Response containing information about certificates status
  - Signed by the server

How is made the verification of a signature? Look at figure 24.

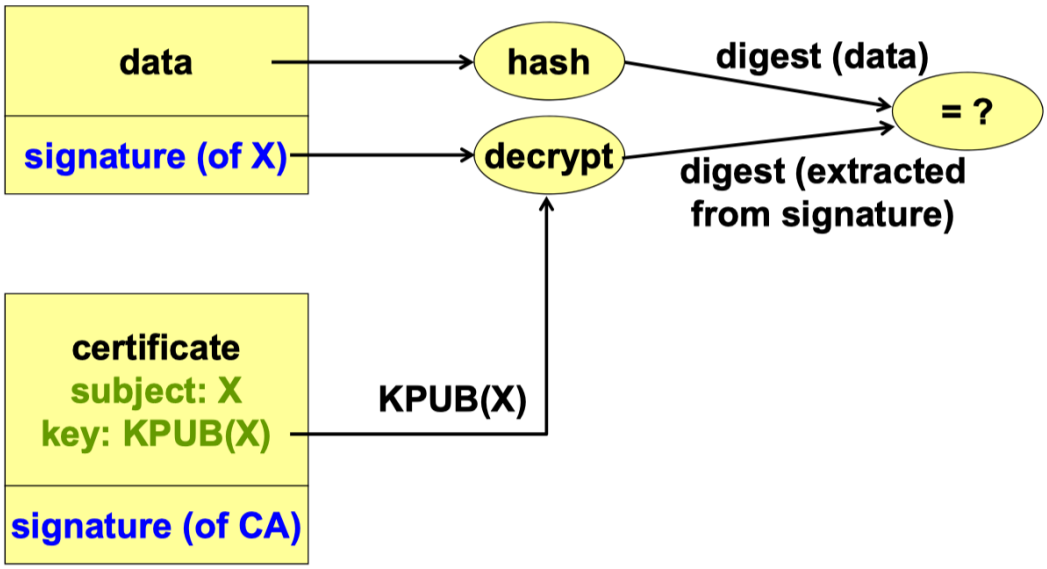

Figure 24: Verification of signature/certificate

A hierarchical Infrastructure is necessary to correctly verify the authenticity. At a certain point a TOP-LEVEL CA is needed, like "god". An example is showed in figure 25:

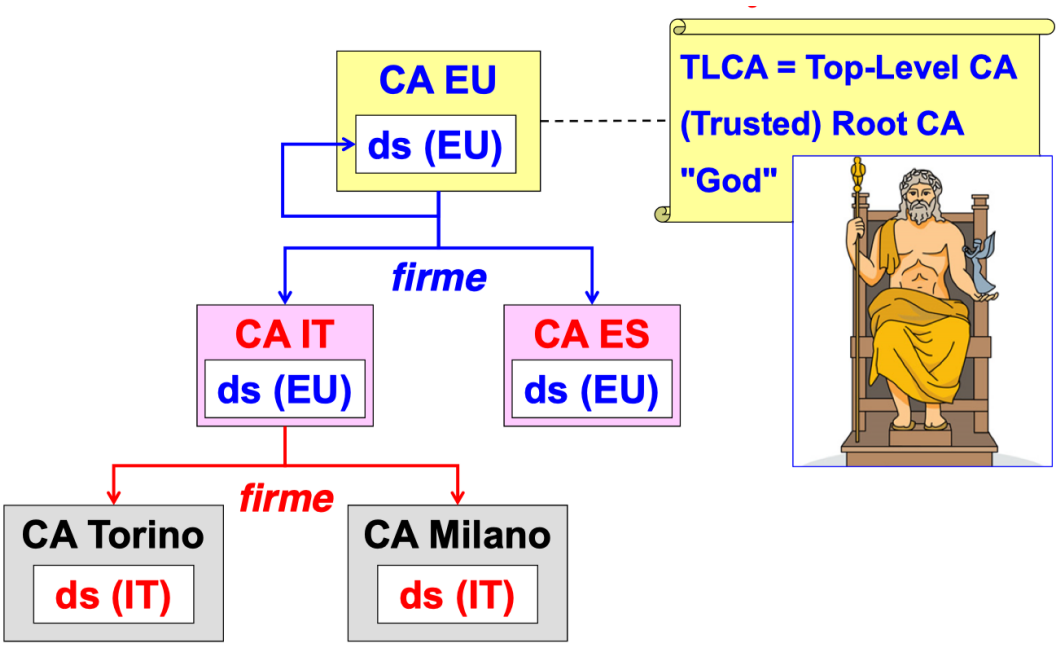

Figure 25: Certification Hierarchy