Politecnico di Torino
Master's Degree in Computer Engineering

Computer system security
lecture notes

Main authors: Nicola Gallo, Loris Gabriele, Luca Ghio, Lorenzo Liotino, Muhammad Ali
Professors: Antonio Lioy, Diana Berbecaru
Academic year: 2014/2015
Version: 1.0.0.0
Date: February 28, 2015

Warning

Chapters 5 (partially), 7 (partially) and 8 have been automatically translated by using Google Translator, and the translation has not been checked by humans.

Acknowledgements

Besides the aforementioned authors, this work may include contributions from related works on WikiAppunti and Wikibooks, therefore thanks also to all the users who have made contributions to lecture notes *Computer system security* and to book *Computer system security*.

About this work

This work is published free of charge. You can download the last version of the PDF document, along with the LaTeX source code, from here: http://lucaghio.webege.com/redirs/c

This work has not been checked in any way by professors and therefore it may include mistakes. If you find any of them, you are invited to directly fix them by yourself by making a commit to the public Git repository or by editing lecture notes *Computer system security* on WikiAppunti, or alternatively you can contact the main authors by sending an e-mail to nicola.gallo90@hotmail.it or artghio@tiscali.it.

License

Contents

Chapter 1

Introduction to the security of ICT systems

ICT security the set of products, services, organization rules and individual behaviours that protect the ICT system of a company

- products: the company should buy proper security products;
- services: products offer security services;
- organization rules: products and services have to be used in a certain way;
- individual behaviours: the behaviour of a single individual may ruin the security of the whole company.

ICT security has the duty to guarantee **C.I.A.** = 'Confidentiality, Integrity, Availability'.

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

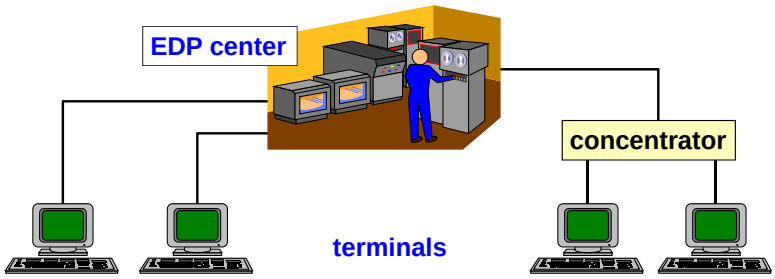
- the ICT system is the safe of our most valuable information;
- ICT security is the equivalent of the locks, combinations and keys required to protect it.

1.1 Why is security needed?

The problem of the security of ICT systems has always existed since the 60s, but nowadays it is becoming more and more important because of three main factors:

- technological: while in the past an attacker needed to physically go to the place where the system he was going to damage was, nowadays an attack to an ICT system can come from everywhere in the world through the network;
- economic: computer attacks can cause significant economic damages, even indirect: for example, an attack to a bank site can be perceived by customers as a risk for their bank accounts;
- strategic: protection of ICT systems controlling public services (aqueducts, railways, atomic power plants, etc.) is of national importance, since many lives depend on their proper functioning.

1.1.1 Traditional paradigm



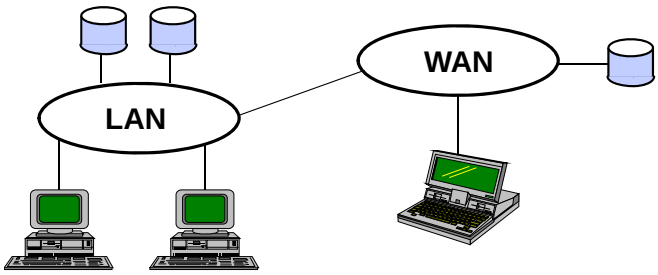
According to the old traditional paradigm, all data and all the computational power are included in a single big computer, called **mainframe**, placed in a room called Electronic Data-Processing (EDP) center, and the access to mainframes is centralized:

- direct access: initially punched cards had to be inserted into a specific reader, which was directly under the control of the system administrator;
- remote access: later distance access was made possible by the appearance of **terminals**, 'stupid' workplaces made up of display and keyboard without any computational or memory capability (only input/output), which were directly connected to the headquarters through end-to-end **leased lines** (multiple terminals in a branch could be aggregated in multiplexing to a single leased line through a concentrator).

This kind of system must be protected:

- at the physical level: the physical infrastructure itself is exposed to attacks:
  - headquarters: the EDP center must be a safe place to prevent an attacker from physically accessing the machine;
  - leased lines: also connection cables must be protected;
- at the logical level: the operating system and applications must include some basic security systems:
  - authentication: a system for user authentication must be implemented (e.g. username and password), so that the mainframe is used only by authorized people;
  - authorization: operations, which every single user is allowed to perform once he is authenticated, must be defined.

1.1.2 Scenario evolution



Nowadays the situation has changed a lot:

- data and processing are distributed ⇒ multiple computers, not only a single centralized one, must be protected;

- terminals are 'intelligent': they have autonomous processing and memory capabilities;
- communications are broadcast through **shared lines** ⇒ as data travels over a channel shared among all nodes, a malicious user could sniff it easily;
- systems are getting more and more complex due to the development of new application paradigms (such as Web, P2P, SMS) ⇒ it is more and more difficult to foresee all kinds of possible attacks.

Often the evolution of technology is not followed by changes in security systems:

- **networks** are insecure:
  - communications are mostly made in clear: it is assumed that nobody else wants to read the transmitted data;
  - Local Area Networks (LAN) operate logically in unicast, but physically in broadcast: it is assumed that who is not the addressee will ignore the received data;
  - in Wide Area Networks (WAN) connections are made through third-party routers: it is assumed that data in transit will not be read or manipulated by who is controlling the router;
- **authentication** is neglected:
  - user authentication is weak, being based simply on username and password: it is assumed that who has entered the password is really its holder;
  - there is no server authentication: normally the user trusts the server;
- **software** contains a lot of bugs, some of which constitute security vulnerabilities exploitable to perform attacks (e.g. DoS attacks):
  - lack of buffer overflow checking (e.g. `gets()` function): if more data than as many as can fit in the buffer are entered, exceeding data will overwrite other memory areas ⇒ the attacker can run arbitrary code injected through a properly manipulated input;
  - storage of sensitive information in cookies: cookies can be read by third parties both during their transit through the network and when residing on the client;
  - storage of passwords in cleartext into a database: they can be read by third parties ⇒ the stored piece of information (e.g. the result of a hash function) should just allow to check whether the password the user has entered is right;
  - implementation of an invented protection system: protection is likely to be inadequate because of design mistakes.

1.1.3 The deep roots of insecurity[1]

'Attack technology is developing in an open-source environment and is evolving rapidly'
Since a lot of attack tools are released under an open source license, they are very easy to be found and improved ⇒ they evolve much more quickly than defensive tools, which often are commercial products.

'defensive strategies are reactionary'
Defensive strategies are reactionary: the trend is to react only after an attack occurred ⇒ defenders are always late as opposed to attackers. For example, an antivirus update is released after a new virus was born and started to infect some computers.

[1] Analysis of document *Consensus Roadmap for Defeating Distributed Denial of Service Attacks* (2000).

'there are tens of thousands – perhaps even millions – of systems with weak security connected to the Internet'
It is not enough that the system you are working on is protected to be safe from possible attacks, but all the other interconnected systems inside the network should be secure as well, otherwise the weakest node could be used as the starting point for an attack to the entire system.

'The explosion in use of the Internet is straining our scarce technical talent. The average level of system administrator technical competence has decreased dramatically in the last 5 years'
Graphical interfaces allow to run automated operations which are not enough when a depth analysis of the problem is required.

'Increasingly complex software is being written by programmers who have no training in writing secure code'
Security must not be the last component to be thought, but it must be considered already at design time: it is important to develop code which contemplates cases of wrong inputs, otherwise an attacker could send unexpected inputs to crash the system. The number of bugs increases exponentially with the number of lines of code (LOC) ⇒ it is better to split the program into multiple components and test them separately to reduce the number of bugs.

'The evolution of attack technology and the deployment of attack tools transcend geography and national boundaries'
The attack origin could be impossible to be located (e.g. the attacker ensured that intermediate computers separate it from the target), or could be within a country lacking legislation for computer crimes.

'The difficulty of criminal investigation of cybercrime coupled with the complexity of international law mean that [...] prosecution of computer crime is unlikely'
The attacker could not be prosecutable if he acted within a country where laws for computer crimes are not well defined. Moreover the high number of attacks discourages the prosecution of all culprits for those crimes.

1.2 Security properties

When designing a security system, you need also to define some abstract properties:

- **authentication**: who are you? (section 1.2.1)
- **non repudiation**: are you the author? (section 1.2.2)
- **authorization**: can you do this? (section 1.2.3)
- **privacy**: has someone access to confidential information? (section 1.2.4)
- **integrity**: has data been manipulated? (section 1.2.5)
- **availability**: is the system working to provide the service in case of unpredictable events (e.g. DoS attacks)?
- **accountability**: after an attack occurred, is it possible to trace to who is the attacker?

availability the property of being accessible and useable upon demand by an authorized entity

accountability the property of a system or system resource that ensures that the actions of a system entity may be traced uniquely to that entity, which can then be held responsible for its actions

All these properties or just some of them could be needed, depending on the kind of security system to be implemented.

1.2.1 Authentication

authentication the process of verifying a claim that a system entity or system resource has a certain attribute value

peer entity authentication the corroboration that a peer entity in an association is the one claimed

data origin authentication the corroboration that the source of data received is as claimed

A security system offers the **authentication** property if it allows to verify a proof which has the purpose to attest the truthfulness of a data or entity attribute:

- **peer authentication**: who is on the other side of the communication channel provides a proof which attests its identity:
  - simple: one of the two peers proves its identity to the other one and not vice versa (e.g. the user provides the server with username and password);
  - mutual: also the other peer proves its identity (e.g. a website provides the user with a digital certificate);
- intrinsic[2] **data authentication**: the data itself contains a proof which attests the truthfulness of one of its attributes:
  - data origin authentication: data contains a proof which attests the truthfulness of its source (e.g. an e-mail contains the sender's digital signature).

Authentication is a different concept from **identification**:

- authentication: it is the result of an electronic procedure (e.g. entering username and password);
- identification: it means to be sure that the login credentials have really been entered by their holder and not by third persons.

(a) ideal *(b) real*

Figure 1.1: Pyramid of security.

Authentication should be the most important property in a security system: if authentication is not strong, all the security measures at the upper layers will not make sense. Keeping track of operations which have been performed (log) makes sense from the security point of view only if there is the certainty of who performed those operations (authentication). However, in reality very economically valuable systems are built upon a very unstable basis, for example relying on an authentication simply based on username and password.

[2] A real-time authentication can not be done when there is not a direct communication channel.

1.2.2 Non repudiation

repudiation denial by one of the entities involved in a communication of having participated in all or part of the communication

A security system offers the **non-repudiation** property if it allows to create a formal proof, acceptable by a court of justice, that gives undeniable evidence of the data creator.

A person or a society could be interested in denying to be the author of data related to illegal actions (e.g. denying to have sent a threatening e-mail).

To achieve this property, many aspects need to be considered: sender authentication, sender identification, data integrity, etc.

1.2.3 Authorization

authorization an approval that is granted to a system entity to access a system resource

access control protection of system resources against unauthorized access

A security system offers the **authorization** property if it allows to define which operations a user is allowed to performed inside the system.

Often authorization coincides with **access control**: when a user tries to access a resource to perform an operation, the authorization module is in charge of verifying whether he has the required permissions.

1.2.4 Privacy

privacy the right of individuals to control or influence what information related to them may be collected and stored and by whom and to whom that information may be disclosed

confidentiality the property that information is not made available or disclosed to unauthorized individuals, entities, or processes

A security system offers the **privacy** (or confidentiality) property if it allows to guarantee that no one can access confidential information without being authorized:

- communication privacy: data crossing a communication channel can be intercepted (e.g. network traffic could be sniffed if it is not encrypted);
- data privacy: data can be stolen from the physical media which it is stored on (e.g. if files are not stored encrypted on a hard disk);
- action privacy: performed actions can be recorded (e.g. an ISP could track website visited by users);
- position privacy: the position can be recorded (e.g. a thief could make a theft when the roomer is away from home).

1.2.5 Integrity

data integrity the property that data has not been altered or destroyed in an unauthorized manner

replay attack an attack in which a valid data transmission is maliciously or fraudulently repeated, either by the originator or by a third party who intercepts the data and retransmits it, possibly as part of a masquerade attack

A security system offers the **integrity** property if it allows to check whether data has been manipulated:

- modification: transmitted or stored data can be changed;

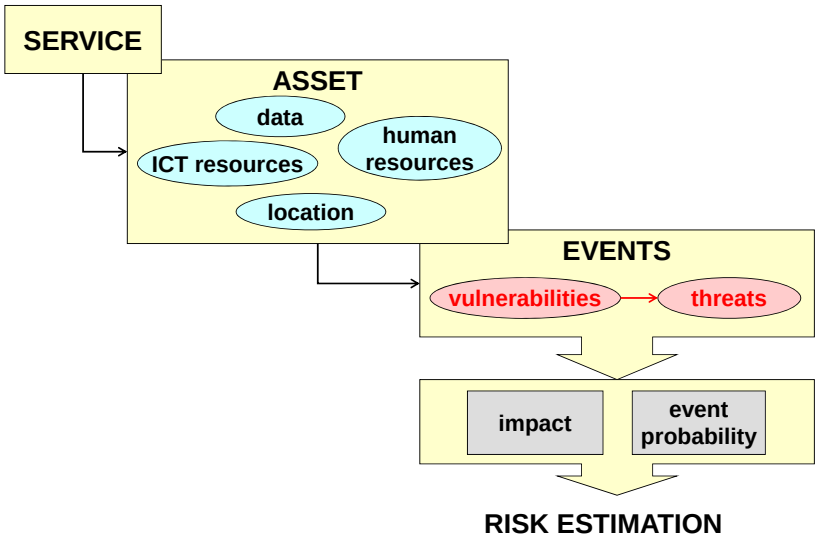
- cancellation: data in transit can be blocked to prevent it from arriving at destination ⇒ the transmitter could require that the receiver sends an acknowledgment to confirm having received it, but the acknowledgment can be faked;
- replay: data in transit, even encrypted, can be copied so as to manage to obtain the effect (e.g. payment) whenever a data copy is transmitted (**replay attack**) ⇒ identification numbers, unchangeable for attackers, should be inserted to detect packet replications.

1.3 Security analysis and management

To guarantee the security of an ICT system, it is required to:

- security analysis: understand which kinds of attacks we have to defend against (section 1.3.1);
- security management: try to prevent the occurrence of these attacks (section 1.3.2).

1.3.1 Security analysis



security service a processing or communication service that is provided by a system to give a specific kind of protection to system resources

asset the set of goods, data and people needed to offer an IT service

vulnerability weakness of an asset

threat intentional or accidental event which can produce the loss of a security property by exploiting a vulnerability

(negative) event occurrence of a threat of type 'accidental event'

attack occurrence of a threat of type 'intentional event'

incident a security event that compromises the integrity, confidentiality, or availability of an information asset

breach an incident that results in the disclosure or potential exposure of data

data disclosure a breach for which it was confirmed that data was actually disclosed (not just exposed) to an unauthorized party

hacker a person who enjoys exploring the details of programmable systems and how to stretch their capabilities, as opposed to most users, who prefer to learn only the minimum necessary

cracker a person who exploits its own computer knowledge to break security on a system

For a security analysis, first of all it is important to define the **service** which is going to be offered to users.

This service is made available through some **assets**:

- ICT resources: CPUs, cables, networks, etc.;
- data: with this definition we mean the actual data, not the disks it is stored on: disks in fact could be completely integral, but data inside them could be compromised by unauthorized access;
- location: it is very important to protect the physical location, because the kinds of attacks which can be performed depend also on where the victim is located;
- human resources: in a company there are some key people which know vital information for the company (e.g. they have the knowledge for running specific systems), and therefore they must be well protected, because if for some reason they decided to leave the company, nobody would be able to run those systems anymore with a consequent loss of money.

Once assets have been outlined, a list of possible **events** have to be made:

- vulnerabilities: each asset is characterized by some vulnerabilities (e.g. location: network cables crossing a street may be damaged by digging works);
- threats: which vulnerabilities can be exploited to perform an attack?

Since the security of a system can not be guaranteed at 100%, it is necessary to focus on higher-priority threats by making a **risk estimation**:

- impact: how long does the service stop working?
- event probability: how many times does a certain threat occur?

Attack origin

To 'build' security, first of all it is necessary to identify where the enemy is:

- outside our organization: a perimeter defense (**firewall**) should be built;
- outside our organization, with the exception of our partners: in addition to the firewall, a safe extranet should be created through a **VPN** solution to secure communications with our partners;
- inside our organization: a protection for the intranet (LAN) should be implemented, but this is contradictory: an intranet is created to make people collaborate, while security usually is an obstacle to collaboration $\Rightarrow$ it is difficult to find a proper tradeoff between favouring collaboration and needing protection;
- everywhere: it is the most realistic option considering the current network condition, where users can connect from everywhere at any time $\Rightarrow$ it is better to put protection **at application level**, covering data as well, rather than in the network.

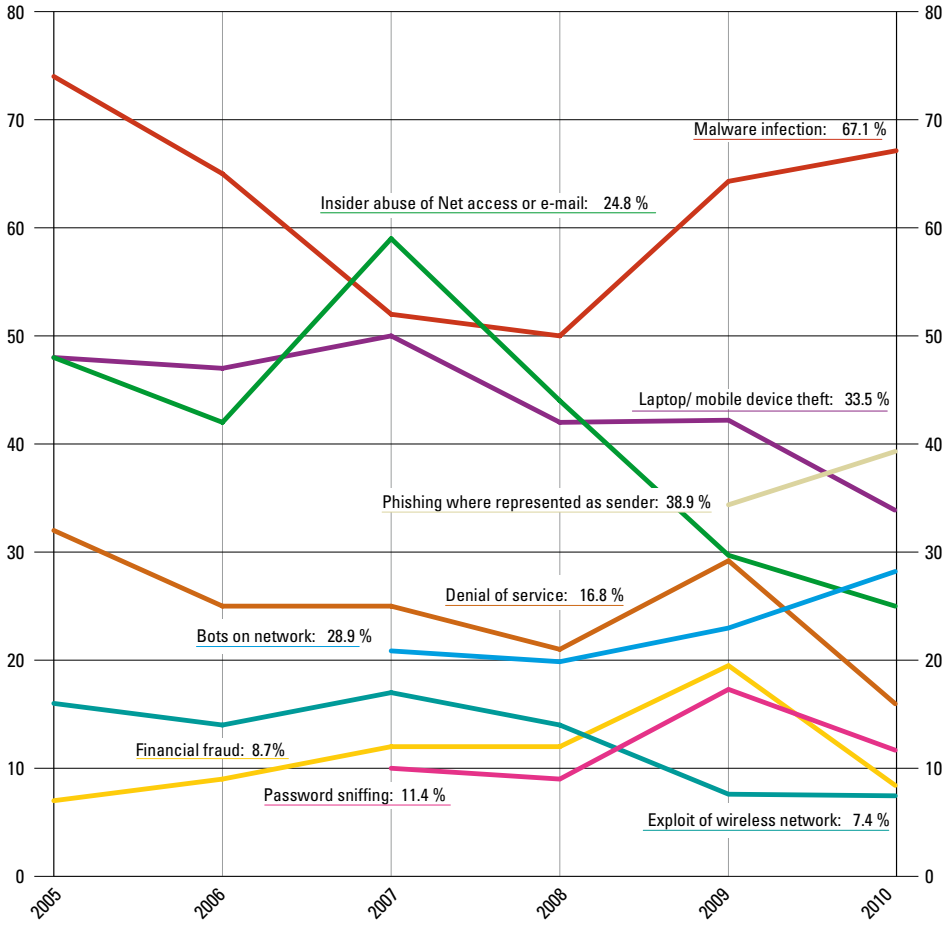


Figure 1.2: Temporal evolution of the main attacks listed in the annual CSI/FBI survey.

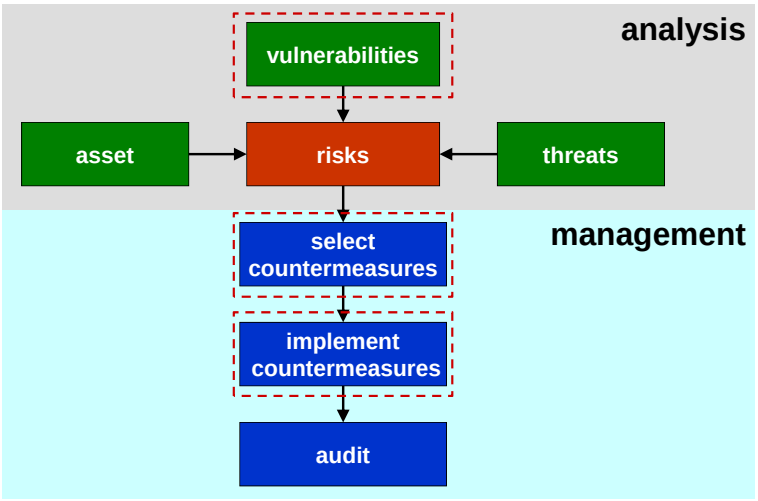
1.3.2 Security management

Once a risk estimation of the system has been performed, it is required to **manage risks**:

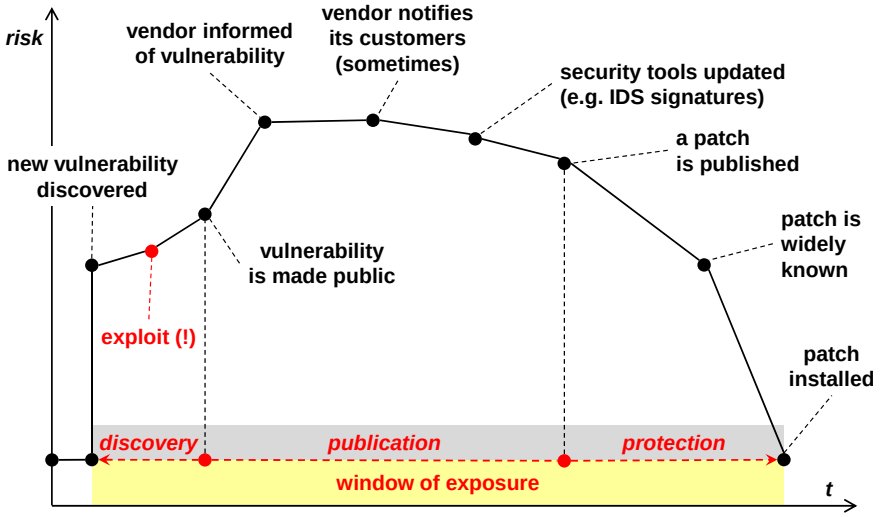
1. select the countermeasures to be adopted in order to block a certain risk, according to costs and performance;
2. implement these countermeasures;
3. audit: check that the countermeasures have been properly implemented and are working as expected (this task must be performed periodically by a person other than the designer).

Security is a process in progress, not a product which can be bought: security flaws are inevitable, therefore the trick is to reduce the risk of exposure regardless of the products or patches:

0. planning: deciding the defensive strategy (e.g. security policies);
1. avoidance: implementing tools which try to avoid attacks (e.g. firewall, VPN, PKI);
2. detection: implementing tools which try to detect if an attacker managed to violate security systems (e.g. Intrusion Detection System [IDS], network monitor);
3. investigation: after every attack, discovering who the attacker is and understanding what did not work to avoid that it will repeat (e.g. forensic analysis, internal audit).
   This last step can bring to a change in the company strategy including new threats or modifying existing countermeasures.



1.3.3 Window of exposure



The **window of exposure** is the period of time in which the system is exposed to a security threat:

1. discovery phase: an attacker exploits an unknown vulnerability:
   (a) an attacker discovers a new vulnerability;
   (b) the attacker develops an attack program, called **exploit**, which exploits this vulnerability;
   (c) the attacker starts performing some attacks, causing the vulnerability to become public;
2. publication phase: people try to limit the damage:
   (a) the victim of the attacks reports the vulnerability to the vendor;
   (b) the vendor notifies all its customers about the existence of this problem;
   (c) customers update their defensive tools (e.g. IDS signatures), waiting for a patch;
   (d) the vendor releases a **patch**;

3. protection phase: customers apply the countermeasure:
   (a) the patch is made available to all customers;
   (b) only when all of them install the patch, the problem can be considered as fixed.

1.4 Basic attacks

- **source address spoofing**: source network addresses in packets are forged (section 1.4.1);
- **packet sniffing**: password and/or confidential data are read by unauthorized third parties (section 1.4.2);
- **DoS** and **DDoS**: service operation is disrupted or limited (sections 1.4.3 and 1.4.4);
- **shadow server**: someone takes the place of a legitimate host (section 1.4.5);
- **connection hijacking**: data is inserted or modified during its transit through the network (section 1.4.6).

1.4.1 Source address spoofing

masquerade a type of threat action whereby an unauthorized entity gains access to a system or performs a malicious act by illegitimately posing as an authorized entity

spoof attempt by an unauthorized entity to gain access to a system by posing as an authorized user

Source address spoofing consists in forging the source network (IP, MAC, etc.) address of a packet.

Attacks The attacker pretends to be someone else (masquerade):

- data forging: data is sent on behalf of someone else;
- unauthorized access: a system is accessed by using someone else's credentials.

Countermeasures Never use authentication based on network addresses.

1.4.2 Packet sniffing

wiretapping an attack that intercepts and accesses information contained in a data flow in a communication system

passive wiretapping only attempts to observe the data flow and gain knowledge of information contained in it

Packet sniffing consists in reading the packets addressed to another network node:

- broadcast networks (e.g. LANs): communications take place through shared channels $\Rightarrow$ it is assumed that who is not the addressee will ignore the received data;
- switching nodes (e.g. switches, routers): each network device typically has a mirror port for traffic analysis.

Attacks The attacker puts its network card in promiscuous mode to intercept anything (passwords, personal data, etc.).

Countermeasures

- non-broadcast networks: in practice they are impossible to make because all networks are based on this principle;
- packet encryption: the attacker can read the data, but can not understand it; however information (addresses and ports) included in the header used by switching nodes should not be encrypted.

1.4.3 DoS

denial of service the prevention of authorized access to a system resource or the delaying of system operations and functions

The **denial-of-service** (DoS) attack consists in keeping a system busy performing useless operations so that it can not provide its services.

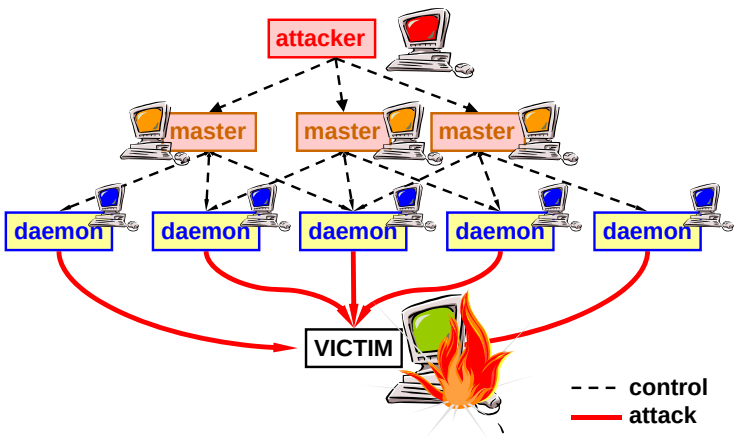
Attacks The attacker prevents the use of a system or a service (availability):

- mail saturation: the mailbox capacity is tried to be saturated so as to make it impossible to receive an important message;
- log saturation: before performing an attack, such a lot of wrong requests are sent to the server that the log storage capacity will run out and the attacker's moves will not be traced;
- ping flooding: ICMP Echo Request messages are sent massively without waiting for replies to them;
- SYN attack: pending TCP connections are opened without sending the ACK closing the three-way handshake, to saturate the tables where active connections are recorded.

Countermeasures No final countermeasure exists, but only palliative remedies based on the quantitative approach to mitigate the effects:

- monitoring: possible anomalies are tried to be detected (e.g. usage of CPU and network resources), although they may be originated from simple malfunctions;
- oversizing: the system is designed to sustain more load than the one normally required, so in case of DoS attack the system will resist for a while, leaving time for identifying the attack source.

1.4.4 DDoS



distributed computing a technique that disperses a single, logically related set of tasks among a group of geographically separate yet cooperating computers

zombie an Internet host computer that has been surreptitiously penetrated by an intruder that installed malicious daemon software to cause the host to operate as an accomplice in attacking other hosts, particularly in distributed attacks that attempt denial of service through flooding

The **distributed denial-of-service** (DDoS) attack multiplies the effect of the DoS attack by putting multiple machines together:

1. the attacker detects the victim;
2. the attacker exploits security flaws to install malware, with self-updating feature, on a lot of remote nodes, which thus become **zombies** (or daemons or malbots), unaware to be accomplices of the attack, and constitute a network called **botnet** (e.g. TrinOO, Tribe Flood Network [TFN], Stacheldraht [= barbed wire]);
3. the attacker elects some zombies as **masters**, nodes with the purpose of synchronizing zombies in the attack (often via encrypted channels);
4. the attacker communicates to masters to launch the attack, then he disconnects from the network so that it will not be possible to trace to him, then leaving the botnet to work autonomously;
5. masters communicate to zombies to launch the attack, all at the same time, against the victim.

1.4.5 Shadow server

shadow server host that manages to show itself (to victims) as a service provider without having the right to do so

The **shadow server** attack consists in replying in an unauthorized manner to requests for a service in place of the legitimate server. This requires:

- packet sniffing: the attacker intercepts the requests for the service;
- address spoofing: the attacker answers the requests as it was the legitimate server;
- DoS: the shadow server must be faster than the real one in replying, so that replies from the real server will arrive later and will be discarded as duplicate packets ⇒ the legitimate server can be slowed down by a DoS attack.

Attacks The victim does not realize that he is communicating with another server:

- providing a wrong service: wrong answers are issued (e.g. a fake DNS server redirects to malicious web servers);
- stealing personal data: the victim provides data to the wrong service.

Countermeasures

- server authentication: you should not trust graphical interfaces, addresses and names.

1.4.6 Connection hijacking

active wiretapping attempts to alter the data or otherwise affect the flow

hijack attack a form of active wiretapping in which the attacker seizes control of a previously established communication association

man-in-the-middle attack a form of active wiretapping attack in which the attacker intercepts and selectively modifies communicated data to masquerade as one or more of the entities involved in a communication association

Connection hijacking (or **data spoofing**) consists in taking control of a communication channel (already set up) and manipulating traffic in transit by inserting, deleting, or modifying packets:

- physical man-in-the-middle: the attacker cuts the cable and physically puts himself in the middle;
- logical man-in-the-middle: the attacker cause a node to send packets to him believing to be sending them to the other party, and then he sends them to the other party (e.g. ARP poisoning: section 5.10.3).

Attacks

- reading, forging and manipulating data exchanged between two parties.

Countermeasures

- authentication: connection hijacking takes control of the communication channel after it has been set up ⇒ peer authentication at the beginning of the communication is not enough, but each single network packet must be authenticated;
- integrity: it allows to check whether a packet has been modified;
- serialization: out-of-sequence or missing packets are not allowed (e.g. `cd /tmp` and `rm *` commands may be swapped losing their harmlessness); sequence numbers must not be modifiable by attackers.

Man-in-the-browser

The man-in-the-middle (MITM) attack is more and more difficult to perform because network channels are more and more protected ⇒ the **man-in-the-browser** (MITB) attack attacks directly user terminals, which instead are less and less protected (e.g. smartphones), by installing software (e.g. keylogger, browser extension) which manipulates traffic before the ideally safe channel.

1.5 Social-engineering techniques

social engineering euphemism for non-technical or low-technology methods, often involving trickery or fraud, that are used to attack information systems

The **social engineering techniques** ask, via e-mail, telephone, or even paper, for the victim user's involuntary participation to the attack action.

Basic problems (non technological)

- low problem understanding: often people do not even realize that a security risk exists;
- psychological pressures: human beings are subject to fallibility (especially when stressed, overloaded, frustrated, etc.) and have a natural tendency to trust;
- complex interfaces/architectures: they can mislead the user and originate erroneous behaviours;
- preformance decrease due to the application of security measures: it brings people to disable security applications becoming possible targets.

1.5.1 Phishing

phishing a technique for attempting to acquire sensitive data, such as bank account numbers, through a fraudulent solicitation in email or on a Web site, in which the perpetrator masquerades as a legitimate business or reputable person

Phishing consists in attracting (via e-mail, instant message, etc.) a network service user to a fake server (shadow server) for:

- acquiring his authentication credentials or other personal information;
- persuading him to install a plug-in or extension which actually is a virus or a Trojan horse.

The 'phishing' term is possibly derived from 'phony fishing': the solicitation usually involves some kind of lure or bait to hook unwary recipients.

Two specialized variants of phishing exist:

- **spear phishing**: several personal data is included to make the message more credible (e.g. known e-mail addresses, real phone numbers, etc.);
- **whaling**: attack targeted to hit important people (e.g. the CEO) which are at the head of the company.

1.5.2 Pharming

pharming attack aimed at redirecting a website's traffic to another, bogus website

Pharming refers to the set of techniques to re-direct a user towards a shadow server (e.g. by changing DNS server addresses on the client):

- direct attack: a vulnerability or misconfiguration is exploited;
- indirect attack: viruses or worms are used.

1.6 Malware attacks

malware any software including malicious code

virus malware which, if run by the user, causes damages in the system; it is not able to propagate to other systems, unless the user transfers, even involuntarily, the infected file

worm malware which saturates CPU, memory or network resources by creating replications of itself, and it is able to propagate to other systems without user intervention

Trojan horse the means through which malwares are usually distributed

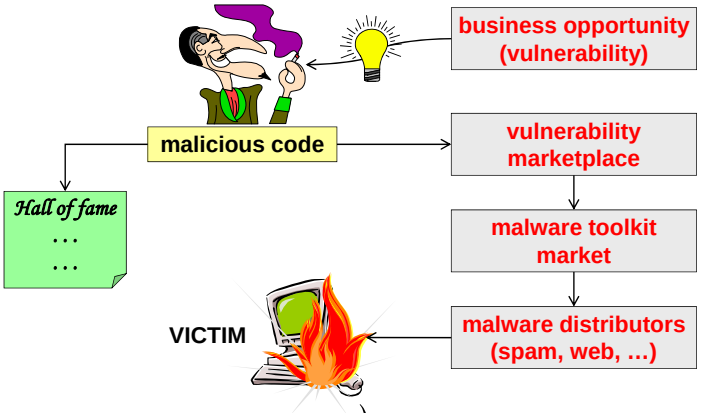
backdoor unauthorized access point which can be used by a malware to bypass the authentication system and enter the system

rootkit set of tools, hidden in the system, to gain system access with administrative privileges (e.g. modified program, library, driver, kernel module, hypervisor)

Malwares are not only a technological issue, but their propagation requires people's complicity (even involuntary):

- users: they are the easiest ones to hit, because they can be hooked by key words (free, urgent, don't open this...) ⇒ users should be sensitized to the security problem and the importance of installing an antivirus;
- system managers: they may configure systems wrongly due to lack of time;
- vendors: they may insert features which violate security (e.g. autorun, trusted areas).

1.6.1 Malware food chain



In the past bad guys were used to develop malware to attack important web sites (e.g. FBI) with the only purpose to gain fame, but nowadays they prefer to gain money by entering the malware food chain:

1. vulnerability discovery: malware always originates from discovery of a new vulnerability in a system not yet known to the public (**zero-day vulnerability**);
2. vulnerability sale: the discovered vulnerabilities are sold at auction on the black market to malware toolkit makers, which need to know as many vulnerabilities as possible to be ahead with respect to antivirus vendors;
3. malware toolkit sale: malware toolkits exploiting vulnerabilities are sold to malware distributors, which manage spam networks or web sites to distribute malware;
4. malware distribution network sale: the final attacker purchases the network from a distributor and uses it to attack the victim.

1.6.2 Zeus

Zeus is a malware discovered in 2007 (it is unknown when it was born) and sold to the highest bidder in 2010. **Zbot** is the name of the zombie botnet on which Zeus is installed (estimation: 3.6 million copies active just in the USA).

Zeus can be used:

- directly by who has control of it:
  - MITB for keylogging: all keys pressed by the user are recorded;
  - form grabbing: data entered into a form are acquired;
- indirectly as a mediator, to load other malware:
  - CryptoLocker ransomware: the victim's hard disk is encrypted, and to get the key for decryption the victim needs to pay some money.

Zeus is very difficult to discover and remove, because it uses a variety of stealth techniques to hide itself in the system.

1.6.3 Stuxnet

Stuxnet is a malware which is a combination of worm + virus for Windows systems. It represents the prototype for a new kind of attack, because it was the first known case of attack against SCADA systems, which are systems used to control industrial plants.

Camouflaging itself as a Windows driver (apparently signed by Microsoft), in 2010 it attacked the IT system who controlled nuclear power plants in Iran by exploiting 4 vulnerabilities existing in Windows services which were enabled without any need:

- 2 zero-day vulnerabilities;
- 1 known vulnerability with available patch (but not installed);
- 1 known vulnerability without any available patch.

As computers were not connected to the Internet, the first infection vector was likely to be a maintenance technician's USB pendrive, and then malware propagated through network disks shared without any protection.

The attack succeeded because of aspects which until that moment had not been considered as security risks, since the system was physically protected by armed guards and was not connected to the Internet:

- lack of standard protections (antivirus, firewall...): the environment was considered trusted;
- lack of periodic Windows updates: the Windows Update service needs to download updates from the Internet;
- misconfiguration: some unnecessary services (MS-RPC, MS-spool, shared network disks) were enabled;
- neglected authentication: the password to access the back-end database was the default one, identical for all systems;
- lack of a validation list: any software could be installed on those computers.

Stuxnet's brothers

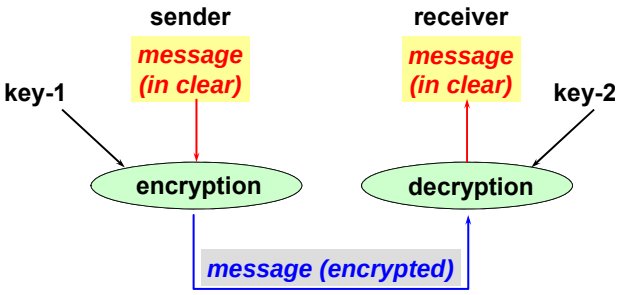
Some Stuxnet's brother malwares sharing the same development platform (tilded) are:

- **Duqu** (2011): it is not a worm or a virus, but carries out reconnaissance and intelligence activities: when installed, it sends out all system information useful to drive an attack;
- **Flame** (2012): it is able to spy systems by recording network traffic, audio, video and keyboard, it spreads through USB pendrives or the network, without ever causing any physical damage, and contains a backdoor to allow remote configuration and update.

Chapter 2

Basics of ICT security

2.1 Basics of cryptography



cryptography the discipline which embodies principles, means, and methods for the transformation of data in order to hide its information content [...]

encryption the cryptographic transformation of data to produce ciphertext

cipher a cryptographic algorithm for encryption and decryption

clear text intelligible data, the semantic content of which is available

cipher text data produced through the use of encipherment; the semantic content of the resulting data is not available

key an input parameter used to vary a transformation function performed by a cryptographic algorithm

Encryption uses a mathematical algorithm which transforms plaintext (or cleartext) data P into ciphertext C through an encryption key K_1, in order to prevent who is not authorized from understanding this data:

$$C = \mathrm{enc}\,(K, P) = \{P\}\,K_1$$

Decryption allows to perform the inverse process through a decryption key K_2:

$$P = \mathrm{dec}\,(K_2, C) = \mathrm{enc}^{-1}\,(K_2, C)$$

Keys K_1 and K_2 are bounded by a mathematical relationship and must always be used coupled: who has not decryption key K_2 can read the encrypted data, but can not understand it because a wrong decryption key brings to a wrong output.

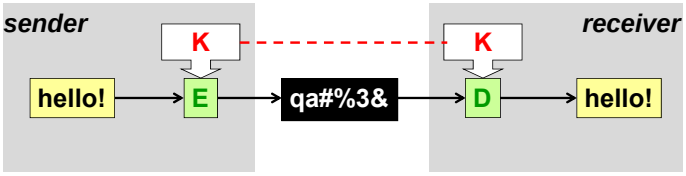
Performance evaluation In general cryptographic performance does not depend on the RAM amount, but essentially on the CPU: architecture, instruction set (instructions already available in hardware), cache size.

Anyway performance is not typically a problem on clients, unless they are overloaded by local applications or are embedded systems with limited computational resources, but it can be a problem on servers and network nodes (e.g. routers) which have to perform cryptographic operations for a lot of clients.

Cryptographic accelerators are cards which implements some cryptographic operations in hardware in order to improve performance:

- generic: they implement just a cryptographic algorithm (e.g. SHA-3);
- specific: they implement a whole security protocol (e.g. SSL, IPsec).

2.1.1 Symmetric cryptography



symmetric cryptography a branch of cryptography in which the algorithms use the same key for both of two counterpart cryptographic operations (e.g., encryption and decryption)

symmetric key a cryptographic key that is used in a symmetric cryptographic algorithm

In **symmetric cryptography** (or **secret-key cryptography**), the symmetric key is a single key K, for both encryption and decryption, shared only between the sender and the receiver:

$$K_1 = K_2 = K \Rightarrow \begin{cases} C = \text{enc}\,(K, P) = \{P\}\,K \\ P = \text{dec}\,(K, C) = \text{enc}^{-1}\,(K, C) \end{cases}$$

Symmetric cryptography requires a lower processing load than asymmetric cryptography $\Rightarrow$ it is particularly suitable for encrypting large amounts of data.

Symmetric algorithms are split into two classes:

- **block algorithms** (e.g. DES, IDEA, RC2, RC5, AES): they split data to be encrypted into blocks of equal size, then they process a block at a time (section 2.4);
- **stream algorithms** (e.g. RC4, SEAL): they are able to work on a data stream, one bit or byte at a time (section 2.5).

2.1.2 Asymmetric cryptography

asymmetric cryptography a modern branch of cryptography (popularly known as "public-key cryptography") in which the algorithms use a pair of keys (a public key and a private key) and use a different component of the pair for each of two counterpart cryptographic operations (e.g., encryption and decryption, or signature creation and signature verification)

asymmetric key a cryptographic key that is used in an asymmetric cryptographic algorithm

key pair a set of mathematically related keys – a public key and a private key – that are used for asymmetric cryptography and are generated in a way that makes it computationally infeasible to derive the private key from knowledge of the public key

public key the publicly disclosable component of a pair of cryptographic keys used for asymmetric cryptography

private key the secret component of a pair of cryptographic keys used for asymmetric cryptography

In **asymmetric cryptography** (or **public-key cryptography**), the asymmetric key is made up of a pair of keys K_{pri} and K_{pub}, distinct based on the way in which they are kept:

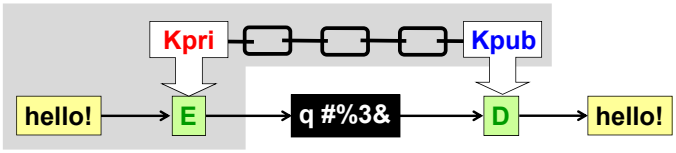
- the **private key** K_{pri} is known only by its owner, and it is kept secret to the rest of the world;
- the **public key** K_{pub} is spread as widely as possible to the rest of the world.

Key characteristics

- interchangeable roles: the keys are not distinct based on their roles, but only based on the way in which they are kept, because each of them can be used for both encryption and decryption;
- reciprocal functionality: data encrypted by a key can be decrypted only with the other one in the same pair.

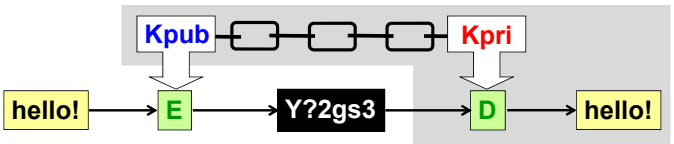
In data transmission from a sender to a receiver, two important security properties can be achieved:

- **origin data authentication**: the sender's asymmetric key is used:



  1. the sender encrypts data by his own private key K_{pri};
  2. the receiver decrypts data by the sender's public key K_{pub}: the output from decryption by the public key is correct only if for decryption the corresponding private key was used ⇒ the receiver has the proof that data was really encrypted by the sender, the only one who was able to use the private key;

- **confidentiality without shared secrets**: the receiver's asymmetric key is used:



  1. the sender encrypts data by the receiver's public key K_{pub};
  2. the receiver decrypts data by his own private key K_{pri}: the output from decryption by the private key is correct only if for decryption the corresponding public key was used ⇒ the sender has the assurance that data will be decrypted only by the receiver, the only one who will be able to use the private key.

Asymmetric cryptography is significantly more complex than symmetric cryptography: it was made possible at the end of the past century thanks to considerable progress in mathematics. Even for current computer systems, asymmetric cryptography has significant computation difficulties, being much slower with respect to symmetric cryptography. For this reason, asymmetric cryptography is suitable to encrypt small amounts of data (a few tens of bits), mainly for:

- digital signature: it exploits origin data authentication (section 2.11.1);
- secret key in-band distribution: it exploits confidentiality without shared secrets (section 2.3.1).

2.2 Strength of a cryptographic algorithm

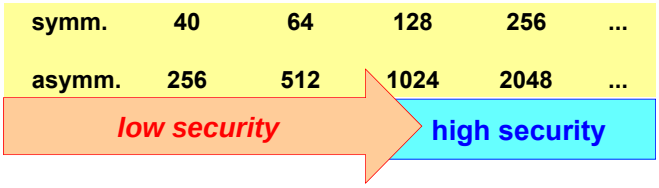


Figure 2.1: The continuous evolution over time of computational power requires longer and longer keys.

strong used to describe a cryptographic algorithm that would require a large amount of computational power to defeat it

key length the number of symbols (usually stated as a number of bits) needed to be able to represent any of the possible values of a cryptographic key

effective key length a measure of strength of a cryptographic algorithm, regardless of actual key length

security by obscurity attempting to maintain or increase security of a system by keeping secret the design or construction of a security mechanism

A cryptographic algorithm is as **strong**er as more difficult it is for a possible attacker to decrypt ciphertext.

The key length should be a tradeoff, based on the computational power available on current systems and the level of secrecy of the encrypted message, between:

- complexity: encryption and decryption should not last too much time;
- security: a brute-force attack should not last too little time.

Kerckhoffs's principle The strength of an algorithm is obtained not by keeping it secret, but by disclosing it so that it will be analyzed as much as possible by the cryptography research community to improve its design. If keys:

- are kept secret: the user should memorize them or keeping them in a safe place;
- are managed only by trusted systems: a malware installed on them could easily steal them;
- are long enough: a brute-force attack should not be feasible, that is it should not last too little time;

then it has no importance that the cryptographic algorithm is kept secret, on the contrary it is better to make it public so that it can be widely studied and its possible weaknesses can be identified.

2.2.1 Strength of symmetric algorithms

Single encryption

If:

- the effective key, N bits long, satisfies the hypotheses of Kerckhoffs's principle;
- the encryption algorithm was well designed;

then the only possible attack is the **brute-force attack** (or exhaustive attack), which requires a number T of trials exponential in the key length N:

$$T = 2^N$$

Even number of encryptions

An encryption algorithm should never be applied an even number of times. In fact, the idea of applying twice the encryption operation, even by two different keys K_1 and K_2:

$$C = \text{enc}\left(K_2, \text{enc}\left(K_1, P\right)\right)$$

is a big mistake: double application of encryption algorithms is subject to a known-plaintext attack, called **meet-in-the-middle**, which allows to find keys K_1 and K_2 by applying twice the brute-force technique $\Rightarrow$ despite the fact that the computation time doubles, the effective key length increases of just one bit, gaining a single 'bit of security':

$$T = 2 \cdot 2^N = 2^{N+1}$$

Assuming to know:

- length N of two keys K_1 and K_2;
- plaintext P;
- ciphertext C;

the attacker can perform the meet-in-the-middle attack in this way:

1. he encrypts plaintext P by all possible encryption keys K_i;
2. he decrypts ciphertext C by all possible decryption keys K_j;
3. he compares outputs to find the output such that:

$$\text{dec}\left(K_2, C\right) = \text{enc}\left(K_1, P\right)$$

4. he easily discards false positive (K_i, K_j) pairs if he known more than one (P, C) pair.

This is why double DES does not exist.

Odd number of encryptions

An encryption algorithms can be applied an odd number of times only if the basic symmetric algorithm is not a group (from the mathematical point of view). If the algorithm is a group, then its strength will not change, because key K always exists such that:

$$\text{enc}\left(K_3, \text{enc}\left(K_2, \text{enc}\left(K_1, P\right)\right)\right) = \text{enc}\left(K, P\right)$$

As DES is not a group, 3DES can be applied in Encrypt-Encrypt-Encrypt (EEE) mode increasing its strength (please refer to section 2.6.2).

2.2.2 Strength of asymmetric algorithms

Length of public keys

Asymmetric keys must be an order of magnitude longer than symmetric ones because:

- the public key carries with itself some information about the private key;
- not all keys are valid, but keys follow well-defined mathematical rules.

The optimal key length has been tested through RSA challenges, whose purpose was to manage to factorize the result as quickly as possible. It has been proved that:

- 512 bits can be attacked in some weeks;
- 1024 bits can be attacked in some months;
- 2048 bits offer a proper security level for several years.

Complexity

Asymmetric cryptographic algorithms are based on problems mathematically difficult to solve for bad people, but easy to apply for good people:

- RSA: the good guy needs to compute the product of two prime numbers, while the bad guy needs to compute the factorization of the result (that is to obtain prime numbers):

$$N = P \cdot Q \Rightarrow \begin{cases} P = {}^{N}/_{Q} \\ Q = {}^{N}/_{P} \end{cases}$$

- DH, DSA: the good guy needs to compute the discrete exponentiation, while the bad guy needs to compute the discrete logarithm of the result (that is to obtain the base and the exponent):

$$\begin{cases} A = g^x \bmod p \\ B = g^y \bmod p \end{cases} \Rightarrow \begin{cases} x = \log_g A \\ y = \log_g B \end{cases}$$

No polynomial algorithms, able to solve bad people's mathematical problems in a reasonable time by modern computers, are currently known:

- factorization: all known algorithms are not polynomial;
- discrete logarithm (often adaptable even for factorization):
  - successive products: they take a time linear in p, and hence exponential in its number of bits;
  - Shor's algorithm: it is polynomial ($\sim O\left(\log^3 N\right)$), is able to solve RSA and DH, but requires a quantum computer;
  - other algorithms (e.g. Pohlig-Hellman, number field sleve): they are better, but are not polynomial.

2.3 Cryptographic key distribution

2.3.1 Secret key distribution

key establishment a procedure that combines the key-generation and key-distribution steps needed to set up or install a secure communication association

key generation a process that creates the sequence of symbols that comprise a cryptographic key

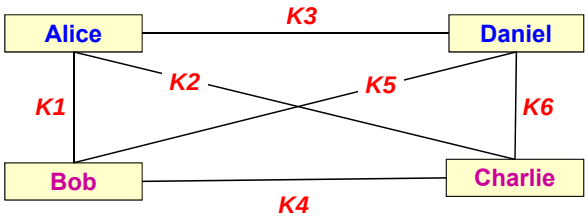


Figure 2.2: Communications among 4 parties require distribution of 6 keys.

key distribution a process that delivers a cryptographic key from the location where it is generated to the locations where it is used in a cryptographic algorithm

out-of-band information transfer using a channel or method that is outside (i.e., separate from or different from) the main channel or normal method

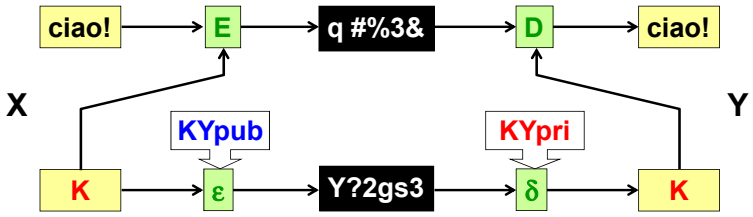
Given N parties which want to communicate with each other in a secure way, the number n of keys needed for a complete private communication grows linearly with the number N of parties to the square:

$$n = \frac{N\,(N-1)}{2} = O\left(N^2\right)$$

Once the secret key is chosen, two strategies exist to deliver it to the other party in a secure way:

- out-of-band (OOB) distribution: the key is transmitted over a different channel than the one normally used for encrypted messages (e.g. normal channel = Internet, alternative channel = telephone);
- in-band distribution: the key is exchanged by means of an asymmetric cryptographic algorithm (page 32);
- key agreement: the key is agreed by means of a key-agreement algorithm (page 33).

Secret key exchange by asymmetric algorithms



key transport a key establishment method by which a secret key is generated by a system entity in a communication association and securely sent to another entity in the association

Confidentiality without shared secrets is often used to communicate in a safe way the secret key chosen for a symmetric algorithm:

1. sender X chooses a key K;
2. X encrypts data by using key K through symmetric algorithm E;
3. X encrypts key K by using receiver Y's public key $K_{Y\,\mathrm{pub}}$;
4. X sends the encrypted key and the encrypted data to Y;

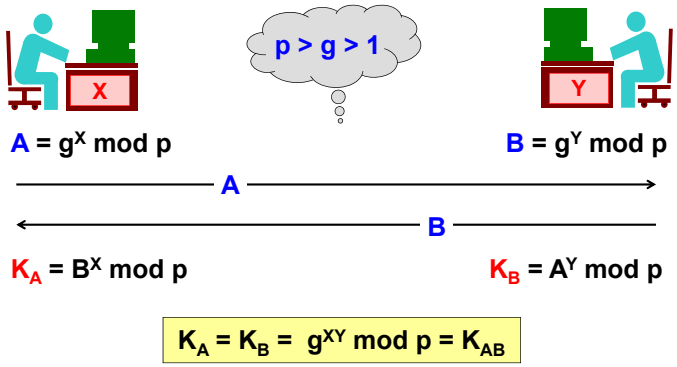
5. Y decrypts key K by using his own private key $K_{Y\,\text{pri}}$;
6. Y decrypts data by using key K through symmetric algorithm D.

In such a scheme there are two chains of transmission and encryption:

- symmetric chain: used to encrypt quickly and transmit a large amount of data;
- asymmetric chain: used to encrypt and transmit a small amount of data, that is the symmetric key.

In practice, this transmission method may not be considered safe, especially for military purposes: the symmetric key is chosen arbitrarily by the sender ⇒ the receiver should trust that the sender did not revealed the key to third parties (e.g. enemies).

Diffie-Hellman



key agreement a method for negotiating a key value on line without transferring the key, even in an encrypted form, e.g., the Diffie-Hellman technique

Diffie-Hellman (DH) was the first algorithm to use the concept of publishing something (numbers p e g), and hence in this sense it is defined as the first public-key algorithm invented.

DH is not a method to distribute the key, but to agree the key:

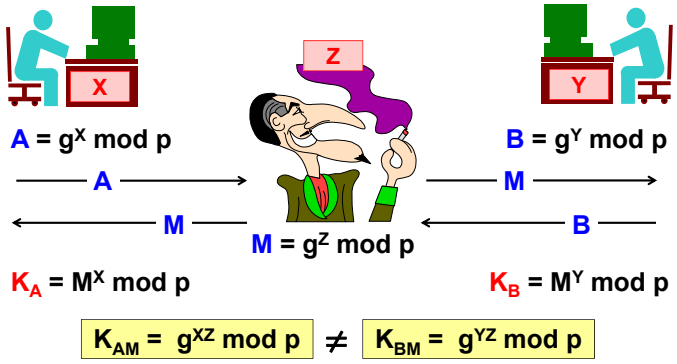
1. sender X and receiver X choose or agree two public integers, a generator g (typically 2, 3, or 5) and a prime number p (large), such that:
$$p > g > 1$$
2. X arbitrarily chooses an integer $x > 0$ (large), and computes:
$$A = g^x \bmod p$$
3. Y arbitrarily chooses an integer $y > 0$ (large), and computes:
$$B = g^y \bmod p$$
4. X sends (publishes) computed value A to Y;
5. Y sends (publishes) computed value B to X;
6. X receives B, and computes:
$$K_A = B^x \bmod p$$

7. Y receives A, and computes:
$$K_B = A^y \bmod p$$

8. due to the properties of exponentiation, values K_A and K_B, computed independently by X and Y, are equal and constitute secret key K_{AB}:
$$\begin{cases} K_A = (g^y)^x \bmod p \\ K_B = (g^x)^y \bmod p \end{cases} \Rightarrow K_A = K_B = g^{xy} \bmod p = K_{AB}$$

Both peers are involved directly in choosing the key ⇒ this method can be used when the peers do not trust each other and do not accept that one of them invents the key: the sender can not have revealed previously to third parties the secret key because it is created at the time of communication itself.

DH is resistant to sniffing attacks: a man in the middle can intercept A and B in transit, which are computed solely based on public data, but he does not know secret data x and y and hence he can not compute the secret key K_{AB}.



Active man-in-the-middle attack However in case attacker Z is able to manipulate data in transit, then he can sniff the traffic between the peers by manipulating the key agreement process:

1. X and Y choose x and y, and compute A and B;
2. Z chooses z and, knowing p and q because public, computes M:
$$\begin{cases} A = g^x \bmod p \\ B = g^y \bmod p \\ M = g^z \bmod p \end{cases}$$
3. X sends computed value A to Y;
4. Z replaces A with M, and sends it to Y;
5. Y sends computed value B to X;
6. Z replaces B with M, and sends it to X;
7. X receives M, and computes:
$$K_A = M^x \bmod p$$
8. X receives M, and computes:
$$K_B = M^y \bmod p$$

9. values K_A and K_B are no longer equal, and moreover they are known to attacker Z:

$$\begin{cases} K_A = (g^z)^x \bmod p \\ K_B = (g^z)^y \bmod p \end{cases} \Rightarrow K_A = g^{xz} \bmod p \neq K_B = g^{yz} \bmod p$$

Attacker Z managed in this way to perform two separate key agreements:

- a key agreement with X, agreeing key K_A;
- a key agreement with Y, agreeing key K_B.

Whenever X sends data to Y, it can be sniffed by Z while the two peers do not realize this:

1. X encrypts data by key K_A and sends it to Y;
2. Z intercepts encrypted data and decrypts it by key K_A;
3. Z can read and even modify data as he wishes;
4. Z encrypts data by key K_B and sends it to Y;
5. Y decrypts received data by key K_B.

This operation appears altogether transparent to sender and receiver: they can not know in any way that beyond the protected channel there is not the counterpart but there is the attacker, the keys which they believe to have agreed are different, and data has been manipulated. Therefore exchanged keys should be protected through a pre-authentication process by using:

- certificates for DH keys;
- authenticated protocols variant of DH (e.g. MQV: invented by Menezes-Qu-Vanstone and patented by CertiCom).

2.3.2 Public key distribution

The distribution of the public key requires some assurances on the binding between the public key and the person's identity:

- authenticity: is the public key actually belonging to who states to owns it?
- integrity: has the public key been subject to modifications after its owner published it?

Therefore the public keys can be distributed in two secure ways:

- out-of-band key exchange: the key is exchanged on an alternative channel, analogously to secret key distribution;
- distribution by means of a public key certificate: it is a digital identity certificate (please refer to section 3.1).

2.4 Symmetric block algorithms

block cipher an encryption algorithm that breaks plain text into fixed-size segments and uses the same key to transform each plaintext segment into a fixed-size segment of cipher text

Two concepts are at the basis of design of modern symmetric block algorithms:

- confusion: it guarantees that a modification to the plaintext will bring to an alteration in the ciphertext which is impossible to predict;
- diffusion: it guarantees that a modification to the plaintext will bring a alteration in the ciphertext which is spread.

XOR operator The ideal confusion operator is the XOR operator, the most suitable binary operator for cryptography specifications: it is able to make an output with the same probability distribution of values 0 and 1 as the input, that is given a random binary input (each bit can be equal to 0 with 50% probability or 1 with 50% probability) the output will be random in the same way ⇒ the output does not suggest any useful information about the input distribution. Moreover XOR is a primitive operation very quick from the hardware point of view, and it is embedded in all CPUs.

Application modes How can a block algorithm be applied to data which has a different size than the one of the basic block?

> **mode of operation** a technique for enhancing the effect of a cryptographic algorithm or adapting the algorithm for an application, such as applying a block cipher to a sequence of data blocks or a data stream

- data > basic block: ECB (sect. 2.4.1), CBC (sect. 2.4.2), IGE (sect. 2.12.2), combined with padding (sect. 2.4.3) or CTS (sect. 2.4.4) when data is not a multiple of the basic block;
- data < basic block (e.g. transmissions over serial or parallel lines): CFB (sect. 2.4.5), OFB (sect. 2.4.6), CTR (sect. 2.4.7).

It is very important to correctly choose the application mode for an algorithm, otherwise you can be attacked regardless the strength of the used algorithm.

2.4.1 ECB

(a) Encryption. *(b) Decryption.*

The **Electronic Code Book** (ECB) mode consists in:

1. splitting the plaintext to be encrypted into basic blocks of fixed size;
2. applying the encryption algorithm to each i-th block, independently of other blocks:

$$C_i = \text{enc}\,(K, P_i)$$

3. concatenating the resulting blocks to produce the ciphertext.

Decryption takes place by reversing the sequence of operations:

$$P_i = \text{enc}^{-1}\,(K, C_i)$$

Advantages

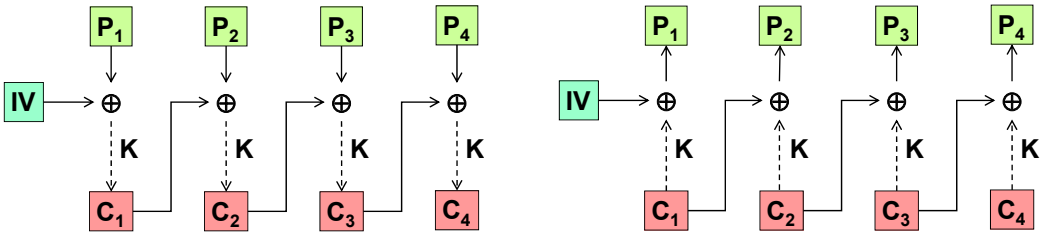
- no error propagation: an error in transmission will cause an error in decryption for a single block without propagating to other blocks;
- CPU parallelization: it is possible to work on multiple blocks at the same time.

Disadvantages When the amount of data is higher than the size of the basic block (there is more than 1 block), encryption of each block is independent of other blocks:

- swapping attacks: encryption is independent of the block position ⇒ any pair of blocks can be swapped, or any block can be moved, preventing this from being discovered;
- known-plaintext attacks: identical blocks are encrypted in the same way ⇒ an attacker could build a database containing outputs from encryptions of a recurrent block (e.g. Word document header) by all the possible keys, then finding the key by comparing each ciphertext block searching for the recurrent block (pattern).

known-plaintext attack a cryptanalysis technique in which the analyst tries to determine the key from knowledge of some plaintext-ciphertext pairs (although the analyst may also have other clues, such as knowing the cryptographic algorithm)

2.4.2 CBC



(a) Encryption. *(b) Decryption.*

initialization value (IV) an input parameter that sets the starting state of a cryptographic algorithm or mode

The **Cipher Block Chaining** (CBC) mode consists in:

1. splitting the plaintext to be encrypted into basic blocks of fixed size;
2. applying the encryption algorithm to each i-th block, XOR-ing the result from the encryption of the $i-1$-th previous block:

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

3. concatenating the resulting blocks to produce the ciphertext.

Decryption takes place by reversing the sequence of operations (XOR is the reverse of itself):

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

Initialization vector As the first block P_1 has not a predecessor, it is XOR-ed to the starting block $C_0 = \text{IV}$. The IV should be made up of random and unpredictable (e.g. not in a sequence) bytes, otherwise the attacker could use databases related to 'easy' IVs to perform the known-plaintext attack to the first block in a short time. The IV can be sent:

- in clear: it should be changed on every transmission, otherwise the attacker would have time to build the database related to that particular IV;
- encrypted by the ECB mode (1 block): the IV can be used for a long time.

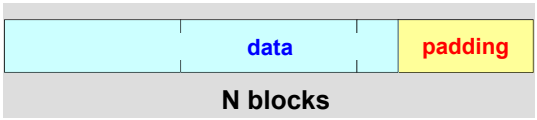
Advantages

- no swapping attacks: swapping two blocks C_i and C_{i+1} will make the decrypted output understandable at blocks P_i, P_{i+1} and P_{i+2};
- no known-plaintext attacks: identical blocks are not encrypted in the same way.

Disadvantages

- error propagation: an error in transmission at block C_i will cause an error in decryption at blocks P_i and P_{i+1};
- no CPU parallelization: cryptographic operations are sequential.

2.4.3 Padding



The **padding** (or aligning, or filling) technique consists in appending L bits to the last N-th block to make data size D a multiple of basic block B, before being able to apply a cryptographic algorithm in ECB or CBC mode:

$$D + L = N \cdot B, \quad 1 \leq L \leq B$$

If the data size is an exact multiple of the basic block, it is required anyhow to insert one entire padding block so as to avoid mistakes in interpreting the last data block ($L = B$).

Padding methods

Which values should padding bits have so that padding can be distinguished from actual data?

- all bytes with null value: `...0x00 0x00 0x00` (if the data length is known a priori or can be obtained, e.g. string terminator `\0`);
- one bit to 1 followed by bits to 0: `...1000000` (original DES);
- one byte with value 128 followed by bytes with null value: `...0x80 0x00 0x00`;
- last byte with value L: `...0x03`:
  - preceded by bytes with null value: `...0x00 0x00 0x03` (Schneier);
  - preceded by bytes with random values: `...0x05 0xf2 0x03` (SSH2);
  - preceded by bytes with value L: `...0x03 0x03 0x03` (SSL/TLS, PKCS #5, PKCS #7);
  - sequence of progressive numbers: `...0x01 0x02 0x03` (IPsec, ESP);
- all bytes with value $L - 1$: `...0x02 0x02 0x02`.

Remarks

- Choosing the padding type for a certain algorithm determines the type of some possible attacks: for example, as SSH2 uses random padding, equal data is encrypted to different ciphertexts, so a padding oracle attack can not be performed.
- The padding types with explicit padding length also offer a (minimal) integrity check: an alteration to data in transmission can be propagated to padding bytes.

2.4.4 CTS

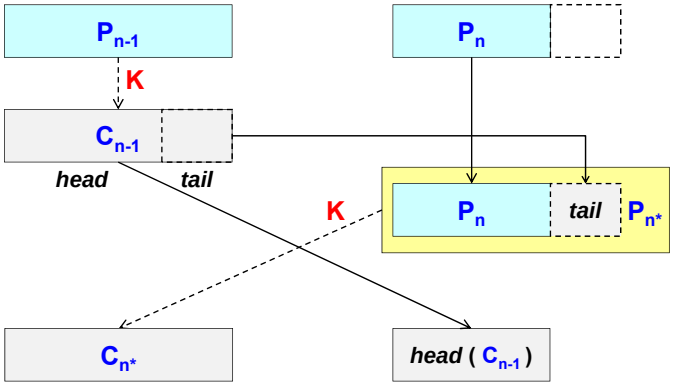


Figure 2.5: Encryption in ECB mode with CTS technique.

The **ciphertext stealing** (CTS) technique allows to use block algorithms in ECB or CBC mode without resorting to padding:

1. penultimate block P_{n-1} is encrypted by using key K, obtaining encrypted block C_{n-1};
2. block C_{n-1} is split into two parts: a head and a tail, the latter exactly of the size which is needed for last block P_n to achieve the size of the basic block;
3. the tail of block C_{n-1} is moved at the end of last block P_n, obtaining block P_{n^*} which now is a multiple of the basic block;
4. P_{n^*} is encrypted by using key K, obtaining block C_{n^*};
5. the head of block C_{n-1} and block C_{n^*} are swapped, because in decryption the tail should be obtained before being able to decrypt block C_{n-1}.

Advantage The size of ciphertext C is equal to the size of plaintext P: a portion of the data itself is used as the padding, avoiding padding overhead $\Rightarrow$ this is useful to encrypt disk sectors: the space on disk can not be exceeded.

Disadvantage The processing time increases because of operations to perform on the last and the penultimate block, when both encrypting and decrypting.

2.4.5 CFB

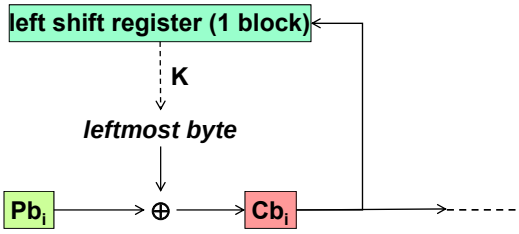


Figure 2.6: Encryption in CFB mode applied to the i-th group of size $N = 8$ bits.

The **Cipher FeedBack** (CFB) mode allows to encrypt a group of N bits at a time, with N variable and lower than the size B of the basic block:

- the block stored in a left shift register is encrypted by using key K;
- from the encryption result, as many leftmost bits are taken as the number of bits in group Pb_i to be encrypted coming from transmission;
- these bits are XOR-ed with group Pb_i, obtaining ciphertext group Cb_i;
- group Cb_i is sent to the receiver as a portion of the ciphertext;
- group Cb_i is inserted into the right side of the shift register.

An IV is required to set the shift register before being able to encrypt the first group.

Disadvantage A transmission error on a group will cause an error in decrypting an entire block (starting from group Cb_i including), until the error will exit the shift register.

2.4.6 OFB

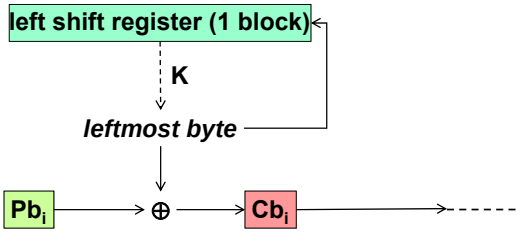


Figure 2.7: Encryption in OFB mode applied to the i-th group of size $N = 8$ bits.

The **Output FeedBack** (OFB) mode differs from the CFB mode due to the fact that the feedback is fed not by ciphertext group Cb_i, but by the group which is XOR-ed.

Advantage A transmission error on a group will cause an error in decrypting just that group.

Disadvantage As the feedback never takes data from outside, values in the shift register are repetitive and predictable in the long run.

2.4.7 CTR

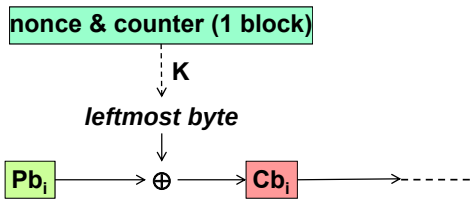


Figure 2.8: Encryption in CTR mode applied to the i-th group of size $N = 8$ bits.

liveness a property of a communication association or a feature of a communication protocol that provides assurance to the recipient of data that the data is being freshly transmitted by its originator, i.e., that the data is not being replayed, by either the originator or a third party, from a previous transmission

nonce a random or non-repeating value that is included in data exchanged by a protocol,

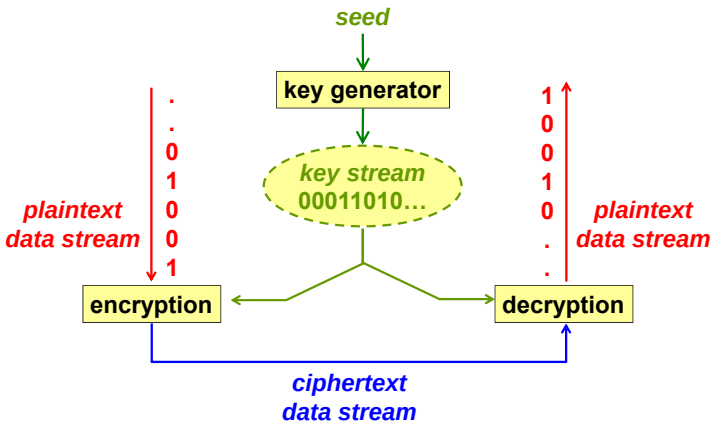
usually for the purpose of guaranteeing liveness and thus detecting and protecting against replay attacks

The **Counter mode** (CTR) differs from the two previous modes due to the fact that the register is no longer shift, but it stores the combination (concatenation, sum, XOR-ing...) between a counter and a nonce (= number used once).

Advantages

- the counter changes the register value upon each group in an unpredictable way;
- the nonce changes the register value upon each transmission in an unpredictable way (the nonce is transmitted as the IV);
- a transmission error on a group will cause an error in decrypting just that group (like the OFB mode);
- CPU parallelization: any group can be encrypted without having to encrypt previous groups (random access).

2.5 Symmetric stream algorithms



stream cipher an encryption algorithm that breaks plain text into a stream of successive elements (usually, bits) and encrypts the n-th plaintext element with the n-th element of a parallel key stream, thus converting the plaintext stream into a ciphertext stream

one-time pad an encryption algorithm in which the key is a random sequence of symbols and each symbol is used for encryption only one time – i.e., used to encrypt only one plaintext symbol and thus produce only one ciphertext symbol – and a copy of the key is used similarly for decryption

pseudorandom a sequence of values that appears to be random (i.e., unpredictable) but is actually generated by a deterministic algorithm

seed a value that is an input to a pseudorandom number generator

A **stream symmetric algorithm** works on a stream of data without requiring to split them into blocks: each bit in the plaintext data stream is matched with a bit in the key flow, and these bits are combined together in some way (typically through the XOR operator) to get the ciphertext data stream.

The key stream must be the same at both encryption and decryption time:

- one-time pad: ideally the key must be really random, but the key is as long as the message to be encrypted ⇒ a key of this size can not be shared with the receiver in a secure way;
- pseudo-random key: real algorithms use pseudo-random key generators, synchronized between the sender and the receiver, and only the seed, from which the sequence is generated, is shared:
  - the key generator should be public, according to Kerckhoffs's principle;
  - the seed should be secret (shared in a secure way) and unpredictable (e.g. not in a sequence).

Advantage Stream algorithms are suitable for encrypting data within a computer (e.g. for encrypting disk backups):

- reliable transmission: transmission takes place via the bus;
- performance: the algorithm is extremely fast, because cryptographic operations consist in simple XOR operations.

Disadvantage Stream algorithms are not suitable for encrypting data transmitted through the network:

- unreliable transmission: multimedia streams flow via UDP, where packets may go lost ⇒ losing a bit during transmission (or its cancellation by an attacker) will cause the ciphertext stream and the key stream to desynchronize at the receiver side, making thus impossible to perform a proper decryption of the message from that bit on;
- packetization: as data is already split into packets, block algorithms are more suitable.

2.6 Main symmetric algorithms

	key length	**block size**	**notes**	**sect.**
DES	56 bits	64 bits	obsolete	2.6.1
3DES	112 bits	64 bits	2 keys, 56/112-bit strength	2.6.2
	168 bits	64 bits	3 keys, 112-bit strength	
IDEA	128 bits	64 bits		2.6.3
RC2	8 to 1024 bits	64 bits	usually $K = 64$ bits	2.6.4
RC4	variable	stream	secret	
RC5	0 to 2048 bits	1 to 256 bits	optimal when $B = 2W$	2.6.5
AES	128, 192 or 256 bits	128 bits	alias Rjindael	2.6.6

2.6.1 DES

The **Data Encryption Standard** (DES) algorithm was designed in the 70s and it had been the standard of the American federal government until 1999.

DES is thought to be hardware-efficient, because it only requires primitives implemented in every CPU:

- XOR;
- shift;
- permutation (one clock tick is enough to perform all needed permutations).

As, since the algorithm was invented, processor computational capabilities have been growing enormously, the brute-force attack is becoming more and more feasible for modern processors: DES is not an intrinsically weak algorithm, but the problem is that the used key is too short: the actual key is totally 64 bits long, but the effective key is just 56 bits long because one parity bit is inserted every 7 bits $\Rightarrow$ the strength T to the brute-force attack is equal to 2^{56}.

In the late 90s cryptographer Ron Rivest, to prove DES vulnerability in favour of his own algorithms, gave life to a series of competitions with prizes awarded to who was able to decrypt some messages which he had encrypted by DES and published on the Web. For DES Challenge III (1998), EFF even built a special-purpose system, called DEEP CRACK, which was able to decrypt a generic DES message within one week, even though with the following limitations:

- the kind of original data (e.g. ASCII) should be known in order to be able to recognize whether the output is good;
- the machine can not decrypt 3DES messages: 3 DEEP CRACKs are not enough because 3DES is a non-linear algorithm.

After 1999 DES was defined as insecure, and IETF changed all RFCs advising not to use DES and suggesting to temporarily use 3DES as a buffer solution.

2.6.2 3DES

The **Triple DES** (3DES) algorithm is nothing more than the repeated application of DES.

Usually the algorithm is used in Encrypt-Decrypt-Encrypt (EDE) mode:

$$C' = \text{enc}\,(K_1, P) \Rightarrow C'' = \text{dec}\,(K_2, C') \Rightarrow C = \text{enc}\,(K_3, C'')$$

1. plaintext data is encrypted by using key K_1, obtaining ciphertext C';
2. ciphertext C' is decrypted by using key K_2, different from key K_1, obtaining ciphertext C'';
3. ciphertext C'' is encrypted by using key K_3, different from key K_2, obtaining ciphertext C.

2 or 3 keys, each one 56 bits long, can be used:

- $K_1 = K_3$: effective key K_{eq} would be $56 \cdot 2 = 112$ bits long, but if the attacker has 2^{59} bytes of available memory, by performing a pre-computation the algorithm strength will decrease to 56 bits (like plain DES);
- $K_1 \neq K_3$: effective key K_{eq} is always $56 \cdot 3 = 112$ bits long.

3DES in EDE mode, unlike the EEE mode, keeps compatibility with DES: if a single key is used ($K_1 = K_2 = K_3$), the final output will be equivalent to the output of plain DES $\Rightarrow$ a single hardware circuit can implement both DES and 3DES.

2.6.3 IDEA

DES was free but, having been developed also by American secret services (NSA), it was suspected to include backdoors $\Rightarrow$ despite the fact that DES was the standard of the American federal government, other cryptography algorithms came to light.

International Data Encryption Algorithm (IDEA) is a patented algorithm but with low royalties (for non-commercial use it was free), owned by Swiss company ASCOM AG. It had been famous until the 2000s because it was the algorithm used by software Pretty Good Privacy (PGP).

It solves the main problem of DES, that is the key length: in fact it uses a 128-bit-long key.

IDEA is especially suitable to be implemented in software, running on 16-bit CPUs, because it requires the following operations:

- XOR;
- addition modulo 16;
- multiplication modulo $2^{16} + 1$ (primitive of CISC CPUs).

2.6.4 RC2, RC4

The **RC2** and **RC4** algorithms were developed by cryptographer Ron Rivest, and are proprietary by his company RSA but are not patented. RC2 was published as an RFC by Rivest himself to be a candidate for AES competition, while RC4 is currently secret (even though it was reverse engineered and unofficial libraries exist such as ARCFOUR).

RC2 and RC4 are respectively 3 and 10 times faster than DES; moreover both of them are implemented at software level. RC2 works on data blocks, while RC4 is stream. Both use variable-length keys.

2.6.5 RC5

The **RC5** algorithm, still developed by Rivest, was published since the beginning as an RFC (Rivest had been payed to make an algorithm specifically for Wireless Application Protocol [WAP]).

A lot of parameters, such as the key length and the basic block size, are variable, but the algorithm works best when block size B is equal to twice a word in the CPU it is running on: $B = 2W$ (**adaptive algorithm**).

It is suitable to be implemented in hardware and especially in software, because it uses the following operations:

- shift;
- rotate (primitive of CISC CPUs, but it can be obtained by two primitives on RISC CPUs);
- modular addition.

2.6.6 AES

The fact that 3DES applies 3 times DES, which was proved to be insecure, was not inspiring a great sense of security ⇒ the USA government issued an international competitive call to find a new symmetric algorithm, which would have been called **Advanced Encryption Standard** (AES) and would have been characterized by the following:

- key length at least 256 bits;
- basic block size at least 128 bits.

Among 15 candidate and 5 finalist algorithms, in 2000 the Rijndael was elected as the winner because:

- it worked on average pretty well, even though never excellent, in all application environments;
- it was the only one among finalists whose authors were not American ⇒ the American government wanted to demonstrate it did not make any preference.

Even if AES was published in 2001, it is still gradually being adopted: before adopting a new cryptography algorithm, in fact it is better to wait some years so that it is analyzed thoroughly by the cryptography community searching for possible attacks.

2.7 Main asymmetric algorithms

The main asymmetric cryptographic algorithms are:

- **RSA** (Rivest-Shamir-Adleman): it provides both confidentiality without shared secrets and data origin authentication (section 2.7.1);
- **DSA** (Digital Signature Algorithm): it provides data origin authentication only, because it performs a lossy compression which prevents original data from being rebuilt;
- **El-Gamal**: it provides confidentiality without shared secrets only, being based on the same mathematical problem as DSA;
- **DH** (Diffie-Hellman): it is mainly used as a key-agreement mechanism (section 2.3.1).

2.7.1 RSA

RSA is an algorithm invented in the 70s by Rivest, Shamir and Adleman. Its patent, valid only in the USA, expired in 2000.

Key generation

To generate the asymmetric key, it is required to:

1. choose two prime numbers P and Q (large), which must be secret, such that their product, **modulus** N, is greater than plaintext p to be encrypted:
$$N = P \cdot Q < p$$
2. arbitrarily choose **public exponent** $E > 1$ such that:
   - E is less than Euler's totient function φ:
   $$E < \varphi = (P-1)(Q-1)$$
   - E is relatively prime[1] with respect to φ;
3. compute **private exponent** D:
$$D = E^{-1} \bmod \varphi$$
4. the asymmetric key is made up of private key K_{pri} and public key K_{pub} pair:
$$\begin{cases} K_{\text{pri}} = (N, D) \\ K_{\text{pub}} = (N, E) \end{cases}$$

The roles of exponents E and D are interchangeable:[2]

- data origin authentication: the sender's asymmetric key is used:
  - signing: the sender uses his own private key:
  $$c = p^D \bmod N$$

[1] A number is relatively prime with respect to another number if they have no factors in common.

[2] In fact due to the properties of exponentiation:
$$\left(p^D\right)^E \bmod N = \left(p^E\right)^D \bmod N$$

  - signature verification: the receiver uses the sender's public key:

$$p = c^E \bmod N$$

- confidentiality without shared secrets: the receiver's asymmetric key is used:
  - encryption: the sender uses the receiver's public key:

$$c = p^E \bmod N$$

  - decryption: the receiver uses his own private key:

$$p = c^D \bmod N$$

Remarks

- The RSA algorithm can encrypt just a small amount of data, precisely p lower than modulus $N \Rightarrow$ RSA is a sort of block algorithm, where the basic block size depends on chosen modulus N.
- Private exponent D is computed as the inverse of public exponent E, which in modular arithmetics is not a rational number (which would require support for floating-point numbers), but is the set of integers such that:

$$E^{-1} \bmod \varphi = D_x \Leftrightarrow (D_x \cdot E) \bmod \varphi = 1, \quad D_x \in \{D_1, D_2, \ldots\}$$

Performance

The complexity of cryptographic operations depends on the number of bits set to 1 in exponents E and D (when the bit is set to 0, the multiplication is not needed):

- public keys: encryption and signature verification operations are usually fast, because typical exponents E (3, 17, 65537 [Fermat number]) have a lot of bits set to 0;
- private keys: decryption and signature operations are slower, but they can be made 4 times faster by the Chinese Remainder Theorem (CRT), used in PKCS #1, thanks to the equivalence:[3]

$$f(x) \bmod N \sim f(x) \bmod P \;\&\; f(x) \bmod Q$$

Attacks

RSA is strong only if it is used in a proper way:

- wrong choice for public exponents: if a signature, made with a public key whose exponent E has a lot of bits set to 1, is provided, this will cause a high computational overload in signature verification (DoS attack):
- wrong choice for public exponents: if the same encrypted message is sent to several recipients all with $E = 3$, then who reads all messages can discover the plaintext:
  - use Fermat number ($E = 65537$) instead of 3, as suggested by the PKCS #1 v1.5 standard;
  - before encrypting the message, always add 'salt' made up of at least 8-byte-long fresh (= nonce) random padding;
- usage of the same key for signature and encryption: as signature and encryption exploit the same operation (just reversed), if an attacker convinces a user to sign by his private key an encrypted message by his public key, then he will get the original plaintext:

[3] "&" is not a bitwise AND operation.

  - use different RSA keys for encryption and signature, to avoid that the signature is equal to the plaintext;

- usage of the user as he was an 'oracle': techniques exist to recover the plaintext if a user just blindly returns the RSA transformation of the input:
  - apply a specific format to the input before encrypting or signing it, and never accept to encrypt or sign raw data;
  - if the format of the decrypted block is different than the expected one, return a generic error and not the decrypted block;
- usage of CRT: it is easier to attack RSA by fault injection techniques, that is techniques which try to change values to see how the result changes and which errors are generated:
  - if the signature verification fails, just report generically 'invalid signature' with no additional details about the error.

2.8 Elliptic curve cryptography

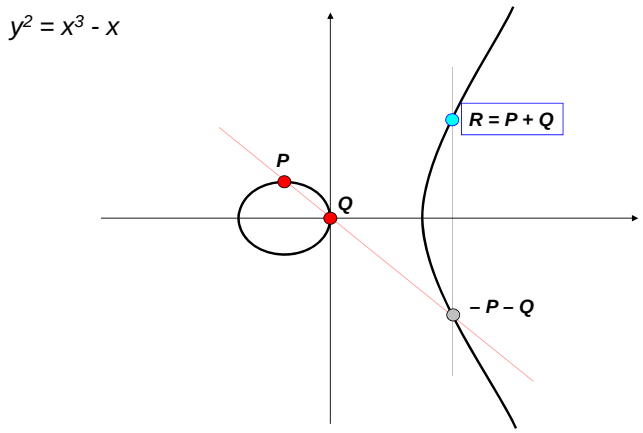


Figure 2.9: Example of elliptic curve used for cryptography.

elliptic curve cryptography (ECC) a type of asymmetric cryptography based on mathematics of groups that are defined by the points on a curve, where the curve is defined by a quadratic equation in a finite field

Elliptic curve cryptography (ECC) uses arithmetics of bidimensional elliptic curves, instead of monodimensional modular arithmetics.

From the conceptual point of view, instead of using numbers (which are points on a straight line) the Cartesian space is taken, and values which are valid are only the points which satisfy the equation of an elliptic curve.

The discrete logarithm problem on a curve is more complex to solve for bad people with respect to the discrete logarithm problem in modular arithmetics ⇒ keys for ECC can be shorter (about 1/10) for a comparable level of security, reducing memory requirements on embedded systems.

Instead for good people all algorithms require just two elementary operations which give points as a result:

- sums of two points;
- multiplications of a point by a scalar.

All problems seen previously are adapted for this kind of arithmetics:

- key distribution: **ECIES** (Elliptic Curve Integrated Encryption Scheme): it generates the key by operations on the curve;
- digital signature: **ECDSA** (Elliptic Curve DSA): the signature is a pair of scalars derived from digest and operations on the curve;
- key agreement: **ECDH** (Elliptic Curve DH): section 2.8.1;
- authenticated key agreement: **ECMQV** (Elliptic Curve MQV).

The American government suggests to use elliptic curve cryptography for key agreement and digital signature for 'secret' and 'top-secret' information.

2.8.1 ECDH

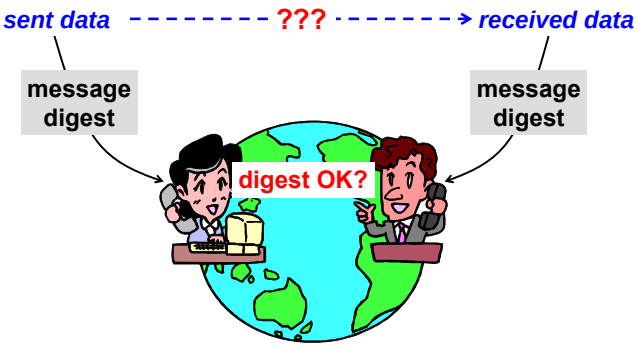
Elliptic Curve Diffie-Hellman (ECDH) is used to agree a secret key K_{AB} analogously as the classical DH:

1. X and Y select the same elliptic curve and a point G of its;
2. X arbitrarily chooses a number x, and computes $A = x \cdot G$;
3. Y arbitrarily chooses a number y, and computes $B = y \cdot G$;
4. X and Y exchange (publish) points A and B;
5. X computes key $K_A = x \cdot Y$;
6. Y computes key $K_B = y \cdot X$;
7. keys K_A and K_B are equal and constitute secret key K_{AB}:

$$K_A = K_B = x \cdot y \cdot G = K_{AB}$$

Also ECDH is resistant to sniffing attacks: the attacker knows the curve and points G, A and B, but he does not know x and y to compute K_{AB}.

2.9 Cryptographic hashing



hash function a function H that maps an arbitrary, variable-length bit string, s, into a fixed-length string, $h = H(s)$ [...]

digest a sort of fixed-length 'summary' of data, usually computed by means of a dedicated (=

for security purpose) cryptographic hash algorithm

Encryption is not a sufficient condition to satisfy the integrity requirement: a person who intercepts an encrypted communication can not understand the meaning of messages, but can modify them in an unpredictable way:

- if at the receiver side there is a human being, he can realize the manipulation because the decrypted data does not make sense;
- if at the receiver side there is a computer, not always it can realize the manipulation because the decrypted data is still bits.

To check that the received data is exactly the same as the sent data, the sender and the recipient need to exchange some information over a parallel (out-of-band) communication channel:

- manual: randomly chosen pieces are compared ⇒ there is no 100% assurance and comparing all pieces would take too long;
- cyclic redundancy check (CRC): computed CRC values are compared ⇒ CRC algorithms are not suitable for security: they expect that unpredictable events (such as weather, radiations) damage single equiprobable bits randomly, but an attacker analyzes the algorithm and tries to modify just the bits which are not checked by the algorithm;
- **digest**: computed 'summaries' are compared ⇒ they are thought specifically to detect manipulations performed by attackers.

2.9.1 Cryptographic hash algorithms

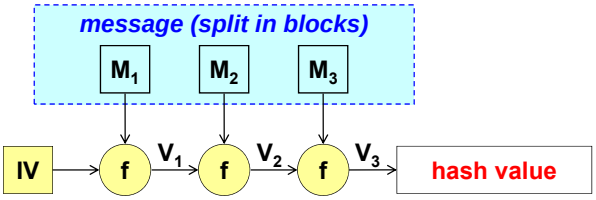


Figure 2.10: General scheme of a hash function.

The digest of a message M is computed in this way:

1. message M, of any length, is split into N blocks $M_1 \dots M_N$;
2. on each block M_k, **hash function** f is applied iteratively, taking as input result V_{k-1} from the previous block (for the first block an initialization vector IV is used):
$$V_k = f\left(V_{k-1}, M_k\right), \quad V_0 = \text{IV}$$
3. digest h of the message is result V_N from last block M_N.

An ideal hash algorithm has the following properties:

- performance: the digest is fast to compute;
- one-way: the digest is impossible to **invert**, that is to obtain the original message from its digest;
- collision-free: it is impossible to obtain a **collision**, that is to obtain the same digest from two different original messages.

Once the algorithm is chosen, the digest length is fixed, independently of the length of input data ⇒ as the digest can be shorter than the original message, the digest computation is a lossy operation, therefore it is definitely impossible to have no probability of **aliasing**: it may happen that two different messages generate the same digest.

2.9.2 Robustness of a hash algorithm

		digest length	block size	known attacks
MD2		128 bits	8 bits	yes
MD4		128 bits	512 bits	yes
MD5		128 bits	512 bits	yes
SHA-1		160 bits	512 bits	partial
SHA-2	SHA-224	224 bits	512 bits	no (?)
	SHA-256	256 bits		
	SHA-384	384 bits	1024 bits	
	SHA-512	512 bits		
SHA-3	SHA3-224	224 bits	1152 bits	no
	SHA3-256	256 bits	1088 bits	
	SHA3-384	384 bits	832 bits	
	SHA3-512	512 bits	576 bits	
RIPEMD		160 bits	512 bits	no

Table 2.1: Main hash algorithms.

An attacker can exploit collisions in order to change transmitted data avoiding that the receiver will realize this: for example, in a transmission through the network where each packet is protected by using a digest, the **birthday attack**[4] can be performed: the attacker waits for two messages to be generated having the same digest (collision), and then he swaps them.

If the algorithm is well-designed, then its robustness depends on the digest length:

- given any two different inputs, probability of aliasing P_A is inversely proportional to 2^N, with N = number of bits in the digest:

$$P_A \propto \frac{1}{2^N}$$

  This merely statistical data however assumes that random bits are changed (**white noise**), while an attacker changes specific bits;

- the only possible attack is the birthday attack, which is unfeasible within a reasonable time if the digest is long enough: when the number of generated messages goes up to $2^{\frac{N}{2}}$, then there is 50% probability of finding at least two different messages with the same digest.

A hash algorithm, which appears good when just invented, may become obsolete because of attacks discovered over time exploiting intrinsic weaknesses related to the algorithm design. If an attack is discovered that enables to have an aliasing probability of 50% by a lower number of inputs with respect to the ones needed for the birthday attack, hence decreasing the time to find a collision, then the hash algorithm is not considered as secure anymore:

- **MD2**, **MD4**, **MD5**: they are currently considered as insecure due to the discovery of definitive attacks;
- **SHA-1**: its robustness is currently doubted due to the discovery of partial attacks;
- **SHA-2**: it is a family of algorithms equal to SHA-1 but with bigger digest lengths ⇒ as design mistakes of SHA-1 remain, SHA-1 attacks theoretically are valid even for SHA-2, even though they are currently not feasible in reasonable times;
- **SHA-3** (alias Keccak): chosen by an international competition for its elegant design, easy analysis and hardware performance, it is still little used because being recent (2008) it has not been analyzed in depth yet;

[4]The name of this attack derives from the **birthday paradox**: if in a room there are at least 23 people, then the probability that two of them were born in the same day is greater than 50%.

- **RIPEMD**: it is considered good because at the moment there are no known attacks, but being poorly used it is has not been analyzed in depth yet (although it was published in 1996).

cryptosystem the set of a cryptographic algorithm and a hash algorithm used to guarantee the security of a certain system

A cryptosystem is **balanced** when the cryptographic algorithm and the digest algorithm have the same strength, that is when the number of bits of the digest is twice the number of bits of the encryption key:

- the SHA-256 hash algorithm (digest = 256 bits) was developed to be used with the AES-128 cryptographic algorithm (key = 128 bits);
- the SHA-512 hash algorithm (digest = 512 bits) was developed to be used with the AES-256 cryptographic algorithm (key = 256 bits).

2.9.3 Digest authentication

The digest should be sent over an out-of-band channel which is trusted by the sender and the recipient, but this channel is not always available. If the digest is sent together with data, an attacker could make some changes on data and then recompute the digest to match the modified data, replacing it to the digest from original data so that the manipulation will not be detectable.

Message Authentication Code (MAC), or **Message Integrity Code** (MIC), is a digest protected so that it can not be changed by third parties preventing the modification from being detected:

- term 'MIC' is mostly used in the telecommunications field, and places emphasis on the fact that it provides integrity of the message, including the digest itself;
- term 'MAC' is mostly used in the application and IT field, and places emphasis on the fact that it provides data authentication of the message, including the digest itself.

To avoid replay attacks, a unique identifier (not necessarily progressive) called **Message IDentifier** (MID) is usually added to every message. MID should be used always bound to a MAC, because without integrity guarantee the ID could be changed as well.

Digest authentication can be based on:

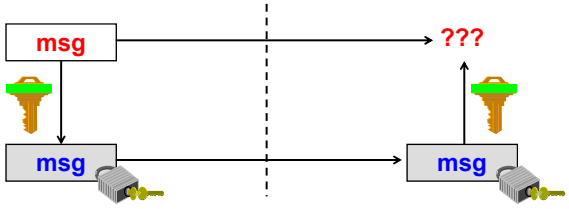
- symmetric cryptography (sect. 2.10): authentication is provided by the shared secret key:
  - being fast it can be also applied to the whole data, guaranteeing the maximum integrity;
  - it is useful just for the receiver: only he is sure that the data has been generated by the sender;
  - it does not provide the non-repudiation property: there is no formal proof that the data has been generated by the sender;
- asymmetric cryptography (sect. 2.11): authentication is provided by the sender's private key:
  - being slow it can be only applied to the digest, with the risk of having collisions;
  - it is useful not only for the receiver: everyone is sure that the data has been generated by the sender;
  - it provides the non-repudiation property: the digital signature is the formal proof that the data has been generated by the sender.

2.10 Authentication based on symmetric cryptography

Authentication is provided by the shared key: being a secret key, it is known only by the sender and the recipient ⇒ only who knows the key can change the MAC so that the modification can not be detected: an attacker can not use the key while being kept secret.

Unlike the digital signature[5], authentication based on symmetric cryptography does not provide the non-repudiation property: since the key is symmetric, it is not possible to distinguish the roles of the two peers, because both of them know the same shared key. This kind of authentication therefore should not be used when the two peers do not trust each other: if there was a litigation, it would not be possible to determine who of the two generated the message.

2.10.1 Authentication by means of symmetric encryption



The MAC consists of the encrypted copy of the data ⇒ the recipient will be able to decrypt the copy and compare it with the received plaintext data.

Security properties

+ integrity: if the plaintext data is changed, the result from the decryption of the copy will be different from the received data;
+ authentication: only who knows the secret key was able to generate an encrypted copy matching the data;
− confidentiality: data is sent in clear;
− non repudiation: which peer generated the data can not be determined.

Advantage full integrity: since this technique does not make use of digests, a modification to any bit in the plaintext will be detected at 100% with no risk of collisions

Disadvantages

- amount of transmitted data: the same data is sent twice;
- computation time: the sender needs to encrypt all data and the receiver needs to decrypt it;
- XOR: the RC4 algorithm can not be used as the symmetric cryptographic algorithm: since encryption is based on XOR, every bit in the plaintext is corresponding to a bit in the ciphertext at the same position ⇒ an attacker may change a bit in the original data and the corresponding bit in the encrypted data.

[5] Please refer to section 2.11.

2.10.2 Authentication by means of CBC-MAC

The MAC consists of the last ciphertext block resulting from applying a symmetric cryptographic block algorithm in CBC mode, with null initialization vector IV (confidentiality is not needed): all ciphertext blocks are thrown away but the last one, which is taken as the MAC ⇒ the recipient will be able to encrypt the received data by the same cryptography algorithm and compare the last resulting ciphertext block with the received ciphertext block.

Data Authentication Algorithm (DAA) uses the CBC-MAC technique with the DES cryptographic algorithm. It was the USA government standard, but today it is no longer considered as secure because of DES vulnerabilities.

Security properties

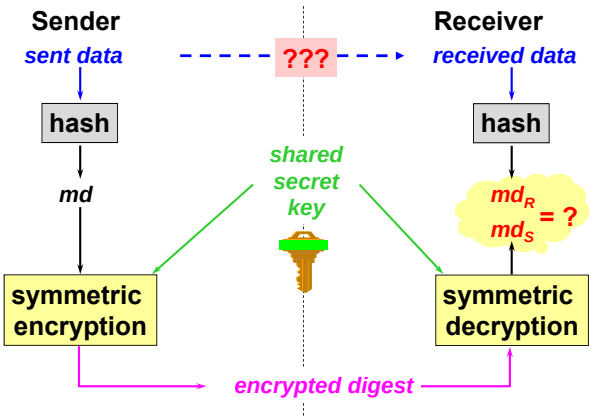
+ integrity: if the plaintext data is changed, the last ciphertext block resulting from the encryption of the received data will be different from the received ciphertext block;
+ authentication: only who knows the secret key was able to generate a ciphertext block matching the data;
− confidentiality: data is sent in clear;
− non repudiation: which peer generated the data can not be determined.

Advantages

- strong integrity: the final block depends on all the previous blocks ⇒ a change will be very likely to affect the last block;
- implementation: a single cryptographic algorithm is enough to guarantee both confidentiality and authentication.

Disadvantage attacks: this algorithm is secure only if the message has a fixed or explicitly-written length, otherwise it would be possible to perform extension attacks (for variable-length messages a variant called **CMAC** needs to be used)

2.10.3 Authentication by means of digest and symmetric encryption



The MAC consists of the encrypted digest of the data:

1. the sender sends the plaintext data to the receiver;
2. the sender computes digest md_S by a cryptographic hash algorithm;

3. the sender encrypts digest md_S by a symmetric cryptographic algorithm by using the shared secret key;
4. the sender sends the encrypted digest to the receiver;
5. the receiver computes digest md_R on the received data by the same cryptographic hash algorithm;
6. the receiver decrypts the received digest by the same cryptographic hash algorithm by using the shared secret key;
7. the receiver compares digest md_R computed on the received data with received digest md_S: if they are the same, then the data will be integer (unless a possible collision).

Security properties

+ integrity: if the plaintext data is changed, digest md_R computed on the received data will be different from digest md_S received and decrypted;
+ authentication: only who knows the secret key was able to generate an encrypted digest matching the data;
− confidentiality: data is sent in clear;
− non repudiation: which peer generated the data can not be determined.

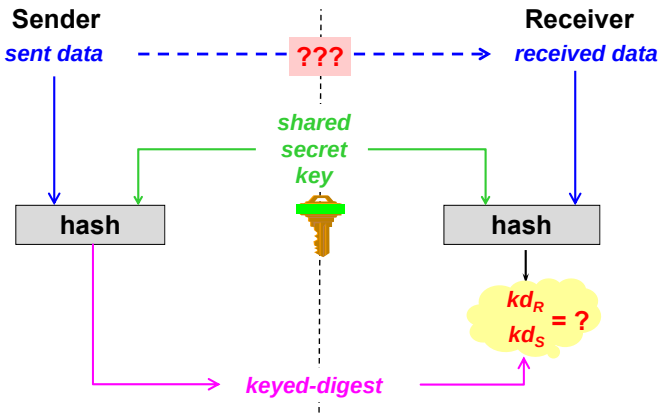
Advantages

- amount of transmitted data: the (encrypted) digest is very small with respect to the data;
- computation time: hash algorithms are fast to run.

Disadvantages

- implementation: two different algorithms are needed, a hash one and a cryptographic one;
- partial integrity: collisions may happen, since it is theoretically impossible to detect all possible modifications.

2.10.4 Authentication by means of keyed-digest



keyed hash a cryptographic hash in which the mapping to a hash result is varied by a second input parameter that is a cryptographic key

The MAC consists of the **keyed-digest** of the data:

1. the sender sends the plaintext data to the receiver;
2. the sender computes keyed-digest kd_S by a cryptographic hash algorithm by using the shared secret key;
3. the sender sends keyed-digest kd_S to the receiver;
4. the receiver computes keyed-digest kd_R on the received data by the same cryptographic hash algorithm by using the shared secret key;
5. the receiver compares keyed-digest kd_R computed on the received data with received keyed-digest md_S: if they are the same, then the data will be integer (unless a possible collision).

Security properties

+ integrity: if the plaintext data is changed, keyed-digest kd_R computed on the received data will be different from received keyed-digest kd_S;
+ authentication: only who knows the secret key was able to generate a keyed-digest matching the data;
− confidentiality: data is sent in clear;
− non repudiation: which peer generated the data can not be determined.

Advantages

- amount of transmitted data: the keyed-digest is very small with respect to the data;
- computation time: hash algorithms are fast to run;
- implementation: a single hash algorithm is enough to guarantee both confidentiality and authentication.

Disadvantage partial integrity: collisions may happen, since it is theoretically impossible to detect all possible modifications

Combination ways

The hash function takes as an input parameter, besides data, the shared secret key too:

1. key K is combined in some way with data M:
$$X = g\left(M, K\right)$$
2. digest kd_M is computed by applying hash function f to result X from the combination:
$$\mathrm{kd}_M = f\left(\mathrm{IV}, X\right) = H\left(X\right)$$

The key has to be combined with data in a proper way, otherwise wrong concatenations make possible some attacks altering the data length:

- prefixed key:
$$\mathrm{kd}_M = f\left(\mathrm{IV}, K \,||\, M\right) = H\left(K \,||\, M\right)$$

  - plaintext message M is changed by postponing any block(s) N:

$$M' = M \,||\, N$$

  - digest kd_M is replaced with digest $\mathrm{kd}_{M'}$ obtained by applying hash function f to block(s) N with digest kd_M as the initialization vector:

$$\mathrm{kd}_{M'} = f\left(\mathrm{kd}_M, N\right) \equiv f\left(\mathrm{IV}, K \,||\, M \,||N\right)$$

- postponed key:

$$\mathrm{kd}_M = f\left(\mathrm{IV}, M \,||\, K\right) = H\left(M \,||\, K\right)$$

  - plaintext message M is changed by prefixing a proper block N:

$$M' = N \,||\, M, \quad N : \mathrm{IV} = f\left(\mathrm{IV}, N\right)$$

  - digest kd_M is not touched:

$$\mathrm{kd}_M = f\left(\mathrm{IV}, M \,||\, K\right) \equiv \mathrm{kd}_{M'} = f\left(\mathrm{IV}, N \,||\, M \,||K\right)$$

Countermeasures

- explicitly inserting the message length into the data itself, so that the data length can not be altered;
- adding the key both before and after the message:

$$\mathrm{kd}_M = H\left(K \,||\, M \,||\, K\right)$$

- using a standard keyed-digest algorithm, which uses a specific hash function by applying it once:
  - **Keyed MD5** (historical): it uses the hash function from the MD5 algorithm:

$$\mathrm{kd}_M = \mathrm{md5}\left(K \,||\, \mathrm{keyfill} \,||\, M \,||\, K \,||\, \mathrm{MD5fill}\right)$$

  - **Keyed SHA1** (version 1 obsolete): it uses the hash function from the SHA1 algorithm:

$$\mathrm{kd}_M = \mathrm{sha1}\left(K \,||\, \mathrm{keyfill} \,||\, M \,||\, K \,||\, \mathrm{SHAfill}\right) \quad \text{(versione 1)}$$

$$\mathrm{kd}_M = \mathrm{sha1}\left(K \,||\, \mathrm{keyfill} \,||\, M \,||\, \mathrm{datafill} \,||\, K \,||\, \mathrm{sha1fill}\right) \quad \text{(versione 2)}$$

- using the **HMAC** standard, which can use any hash function H (e.g. MD5, SHA1) by applying it twice:[6][7]

$$\mathrm{kd}_M = H\left(K' \oplus \mathrm{opad} \,||\, H\left(K' \oplus \mathrm{ipad} \,||\, M\right)\right), \quad \begin{cases} K' = H\left(K\right) & |K| > B \\ K' = K & |K| \leq B \end{cases}$$

  The hash function is applied twice to make the MAC resistant against attacks even if hash function H is not very secure.

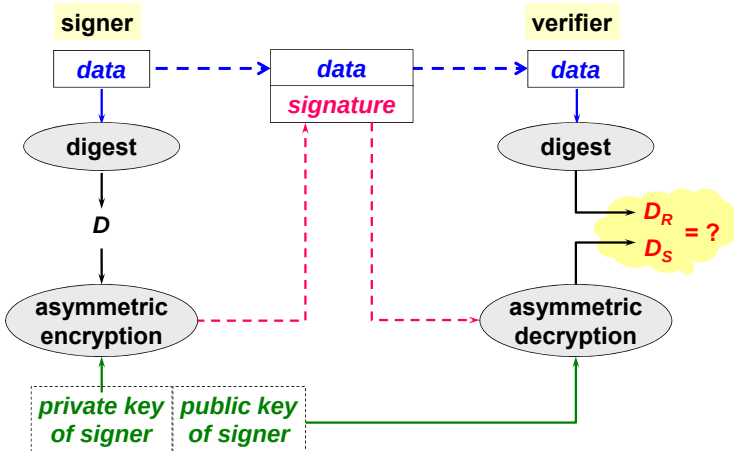
[6] Block size B must be greater than digest length L: $B > L$.
[7] Using keys K shorter than digest length L is deprecated: $|K| \geq L$.

2.11 Authentication based on asymmetric cryptography

Authentication is provided by the sender's private key: being a secret key, it is known only by the sender ⇒ only who knows the key can change the MAC so that the modification can not be detected: an attacker can not use the key while being kept secret.

Authentication based on asymmetric cryptography provides the non-repudiation property: since the key is asymmetric, it is possible to distinguish the roles of the two peers, because only the sender knows his own private key. However, the recipient needs to have the certainty that the used public key really is the sender's one by means of a public key certificate (please refer to section 3.1).

2.11.1 Digital signature



digital signature a value computed with a cryptographic algorithm and associated with a data object in such a way that any recipient of the data can use the signature to verify the data's origin and integrity

Signing The author of an electronic document can sign it by attaching his own **digital signature**:

1. the signer computes digest D by a cryptographic hash algorithm;
2. the signer encrypts digest D by an asymmetric cryptographic algorithm by using his own private key, obtaining the digital signature;
3. the signer binds the digital signature to the data of the document.

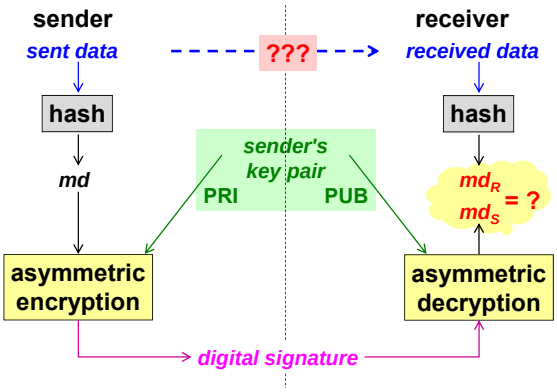
Verification Who reads the electronic document can verify whether who claims to be its author really is by the attached digital signature:

1. the verifier computes digest D_R by the same cryptographic hash algorithm;
2. the verifier decrypts the digital signature by the same asymmetric cryptography algorithm by using the author's public key, obtaining digest D_S;
3. the verifier compares computed digest D_R with decrypted digest D_S: if they are the same, then the data will be integer (unless a possible collision).

The digital signature is stronger than the handwritten signature because it is tightly bound to the data:

- digital signature: on an electronic document it provides authentication and integrity, because it varies depending on the data (infinite digital signatures, one for each different document, can be generated from a single private key);
- handwritten signature: on a paper document it provides authentication but not integrity, because it is always the same regardless of the data ⇒ some text could be added after the document has been signed.

2.11.2 Authentication by means of digital signature



The MAC consists of the digital signature of the data:

- the signer is the sender;
- the verifier is the receiver.

Security properties

\+ integrity: if the plaintext data is changed, digest md_R computed on the received data will be different from digest md_S received and decrypted;

\+ authentication: only who knows the private key was able to generate a digital signature matching the data;

− confidentiality: data is sent in clear;

\+ non repudiation: the sender can not deny to have signed the data, provided his public key was provided by a public key certificate.

2.11.3 PKCS #1

PKCS #1 (Public Key Cryptography Specification #1) is a standard which implements digital signatures, by defining all the primitives for using RSA as the asymmetric cryptographic algorithm in digital signatures:

- conversion and representation of large integers;
- base algorithms for encryption/decryption;
- base algorithms for signing/verification.

These primitives must be used in a proper way to create a standard secure 'cryptographic schema' ⇒ the standard states in a precise way how to perform the following operations:

- encryption/decryption:

  - RSAES-PKCS1 version 1.5: old encryption base schema which has been attacked;
  - RSAES-OAEP (Optimal Asymmetric Encryption Padding): the currently secure encryption schema where an optimal padding is specified, because a bad choice of padding could lead to attacks;

- signing/verification: (these schemas are called **with appendix** because what is encrypted is not the data but its 'summary' [digest])
  - RSASSA-PKCS1 version 1.5: old signature base schema which has been attacked;
  - RSASSA-PSS (Probabilistic Signature Scheme): the currently secure signature schema.

2.11.4 Attacks to digital signature

In case of attack, it is not possible to determine the attacker's target: if the verification of the digital signature fails, that is digests md_R and md_S are not the same, this can be due to three cases indistinguishable from each other:

- the attacker could have altered the data $\Rightarrow$ digest md_R will not be correct;
- the attacker could have altered the digital signature $\Rightarrow$ digest md_S will not be correct;
- the attacker could have claimed to be the legitimate sender $\Rightarrow$ the legitimate sender's public key will not match the attacker's private key.
  The sender's public key must not be provided by the sender, but must be provided by a trusted external entity through a public key certificate, otherwise the attacker could pass his own public key off as the legitimate sender's public key (please refer to section 3.1).

The hash function should have the following properties:

- resistant to collisions: different pieces of data should result in different signatures, otherwise the same signature could be used for other documents;
- difficult to invert: if the hash function was invertible, it would be possible to create fake digital signatures without knowing the sender's private key:
  1. the attacker chooses a digital signature S randomly;
  2. the attacker encrypts digital signature S by using the legitimate sender's public key, obtaining digest R;
  3. the attacker inverts digest R, obtaining data $X = h^{-1}(R)$;
  4. the attacker sends data X and digital signature S, pretending to be the legitimate sender;
  5. the receiver verifies successfully (fake) digital signature S by using the legitimate sender's public key.

2.12 Cryptography with authentication (and integrity)

2.12.1 Separate operations

Data can be simultaneously encrypted and authenticated by performing two separate operations:

- confidentiality: symmetric cryptography with key K_1;
- authentication (and integrity): MAC (keyed-digest) with key K_2.

An improper combination of secure algorithms may lead to an insecure result:

- **authenticate-and-encrypt** (A&E): the MAC is computed on the plaintext and is sent in clear in parallel with the encrypted data (e.g. SSH):

$$\text{enc}\left(p, K_1\right) \,||\, \text{mac}\left(p, K_2\right)$$

  - − DoS attacks: the MAC is computed on the plaintext data ⇒ the received data always needs to be decrypted before being able to compute the digest to compare;
  - − oracle attacks: the MAC is sent in clear ⇒ the MAC could provide indirect information about the plaintext;
  - − security: this combination is insecure, unless it is performed in a single step;

- **authenticate-then-encrypt** (AtE): the MAC is computed on the plaintext and is sent encrypted together with the encrypted data (e.g. SSL, TLS):

$$\text{enc}\left(p \,||\, \text{mac}\left(p, K_2\right), K_1\right)$$

  - − DoS attacks: the MAC is computed on the plaintext data ⇒ the received data always needs to be decrypted before being able to compute both the digests to compare;
  - + oracle attacks: the MAC is sent encrypted ⇒ the attacker can not make any deduction on the plaintext;
  - − security: this combination is secure only with block symmetric algorithms in CBC mode or with stream symmetric algorithms;

- **encrypt-then-authenticate** (EtA): the MAC is computed on the ciphertext and is sent in clear in parallel with the encrypted data (e.g. IPsec):

$$\text{enc}\left(p, K_1\right) \,||\, \text{mac}\left(\text{enc}\left(p, K_1\right), K2\right)$$

  - + DoS attacks: the MAC is computed on the ciphertext ⇒ decryption of the received data can be avoided if the MAC is wrong;
  - + security: this combination is the most secure one, unless implementation mistakes (e.g. initialization vector IV and used cryptographic and hash algorithms must always be included into the MAC).

2.12.2 Authenticated encryption

> **authenticated encryption mechanism** cryptographic technique used to protect the confidentiality and guarantee the origin and integrity of data, and which consists of two component processes: an encryption algorithm and a decryption algorithm

Authenticated encryption (AE) algorithms combine cryptography and keyed-digest operations into a single operation:

- + confidentiality and authentication (and integrity) with a single algorithm and a single key;
- + higher speed;
- + lower risk of implementation mistakes.

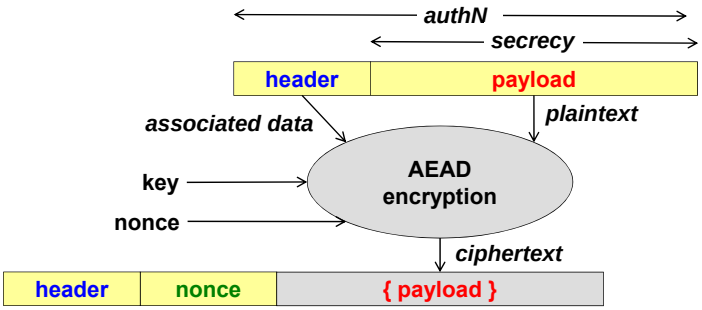
AE is becoming a more and more increasing need at the application layer (e.g. for e-mails). AE algorithms can be:

- AEAD or not: some authenticated pieces of data can be left unencrypted;
- single-pass or double-pass: a double-pass algorithm needs to store results from the first pass for further calculations ⇒ it is at least twice slower than a one-pass algorithm if implemented in software;

- on-line or off-line: data can be taken as they are coming from the wire or first should be all saved.
  Normal encryption schemes are subject to chosen-ciphertext (oracle) attacks if used on-line:

  1. the attacker changes a ciphertext;
  2. the attacker observes whether the receiver reports an error.

AEAD



Only a portion of the network packets needs confidentiality:

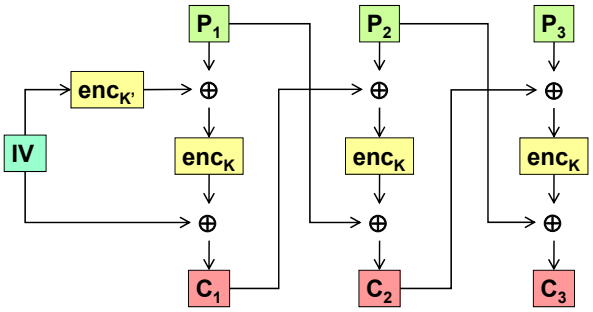
- both header and payload need authentication;
- just the payload needs confidentiality.

Authenticated Encryption with Associated Data (AEAD) algorithms provide authentication without confidentiality for data associated with the plaintext which should be left unencrypted:

- plaintext: the data portion which needs also confidentiality (e.g. payload);
- associated data: the data portion which needs only authentication (e.g. header).

The ciphertext is obtained by combining the plaintext and the associated data together with a key, and typically even a nonce to make the system resistant to replay attacks. The new packet is made up of, besides the ciphertext, the header from the starting packet and the nonce used for encryption.

IGE



Infinite Garble Extension (IGE) is an application mode for block symmetric cryptographic algorithms which adds to the CBC mode the feedback on the plaintext:

- CBC: an error at block C_i will cause an error in decryption just at blocks P_i and P_{i+1};

- IGE: an error at block C_i will cause an error in decryption at block P_i and all the next blocks.

IGE is able to guarantee confidentiality and integrity (and authentication) by using a single algorithm, that is the cryptographic algorithm: integrity is provided by a control block added at the end (e.g. all zeroes, block count): if the last decrypted block is not the expected one, the ciphertext has been changed.

IGE is AE, but is not AEAD because there is not the associated data: all data is encrypted and authenticated.

Other application modes for AE

- **GCM** (Galois/Counter Mode) [AEAD, single-pass, on-line]: it is the most popular one and is parallelizable (used by TLS and OpenSSL);
- **OCB** (Offset Codebook Mode) 2.0 [AEAD, single-pass, on-line]: it is the fastest one, but is little used because initially patented with GPL license;
- **EAX** (Encrypt then Authenticate then X(trans)late) [AEAD, double-pass, on-line]: it is slow but lightweight because it only uses one encryption algorithm ⇒ optimal for limited systems:
  - EAX-prime (EAX′): a more lightweight version of EAX which, since sacrificing some encryption steps and some memory bytes, has been proved as being attackable (used for network transmission of electronic measures, e.g. Enel power meters);
- **CCM** (CTR mode with CMC-MAC) [double-pass, off-line]: it is the slowest one (used in 802.11i wireless networks):
  - CCM*: is is a modified version of CCM which can be configured into the authentication-only or encryption-only optional modes depending on the operating context (used by ZingBee, communication standard for short-range networks).

International competition CAESAR (2013-2017) is ongoing to select an AE algorithm.

2.13 USA cryptographic policy

Until 1996, export of cryptographic material, both in a software and a hardware form, outside the US boundaries was subject to the same restrictions as nuclear material: cryptographic products (e.g. web browsers) could not be exported, unless their protection level was very low (e.g. symmetric keys up to 40 bits[8], asymmetric keys up to 512 bits).

However USA companies were losing market shares abroad because of the poor security offered by their products.

2.13.1 **Key escrow** (December 1996)

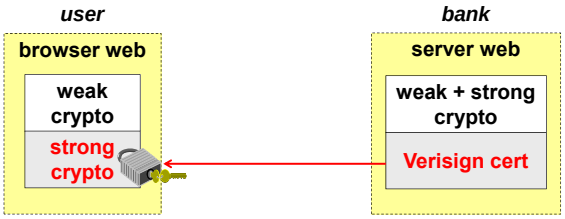
key escrow possibility to recover a key even without the owner's consent

US companies were allowed to export semi-robust cryptographic products (e.g. symmetric keys up to 56 bits), under the condition that they embedded **key escrow** features: a key could be recovered in some way by the American government if needed, even without the owner's consent.

Example Lotus Notes 4.x used 64-bit-long symmetric keys, but in its international version 24 of them were encrypted by the public key of the US intelligence agency (NSA) ⇒ the American government had to guess just 40 bits in order to recover the symmetric key.

[8] In September 1998 the maximum length for symmetric keys was increased to 56 bits, that is the length of the effective key of the DES algorithm which just in that time was starting to be considered as insecure.

2.13.2 Step-up (gated) cryptography (June 1997)



The US government granted permission to export secure web servers provided used by foreign branches of USA companies or in financial environment.[9] The permission was granted only to the entities which purchased a public key certificate issued by VeriSign, which was in charge of checking the real usage of the web server: when a USA bank bought a VeriSign certificate, the web server in its foreign branch could therefore include the strong part of cryptography too.

On the client side, however, the international versions of the web browsers developed by US companies included both the strong part and the weak part of cryptography:

- websites without VeriSign certificate: the browser used the weak part of cryptography, even though the site supported a stronger cryptography;
- websites with VeriSign certificate: the browser received the VeriSign certificate and enabled (**step-up**) the strong part of cryptography for browsing on that specific website.

2.13.3 Key recovery

> **key recovery** mechanisms and processes that allow authorized parties to retrieve the cryptographic key used for data confidentiality

Later USA introduced the permission to export products with strong cryptography (e.g. 128 bits for symmetric keys), provided that they embedded **key recovery** features: when a key was generated, it was stored into one of the four centers authorized by the USA government, encrypted by the public key of the recovery center:

\+ the user could recover the key in case of loss;

− the American government could recover the key for example in case of suspected terrorism.

2.13.4 (Partial) liberalization (January 2000)

US products could at last be exported off-the-shelf, that is without the need for authorizations, provided that one of the following conditions was followed:

- the source code is freely available on the Internet (open source), or
- the product passed a 'one-time review' (NSA is suspected to insert backdoors into the source code).

[9] In September 1998 the permission was extended to insurance and health institutions.

Chapter 3

The X.509 standard, PKI and electronic documents

3.1 Public key certificate

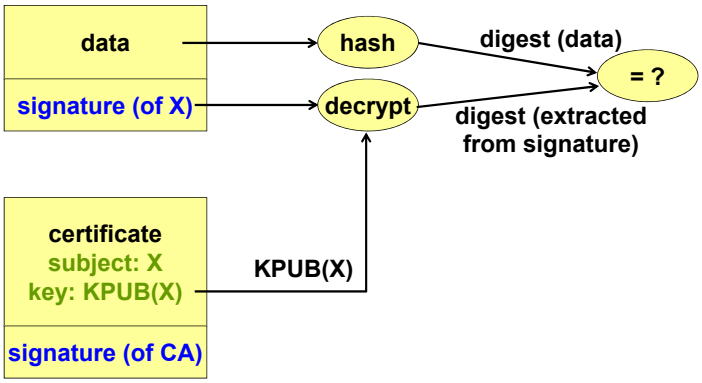


Figure 3.1: Verification of a digital signature by means of a public key certificate.

public key certificate a data structure used to securely bind a public key to some attributes

Public key certificates enable to distribute public keys to use digital signature in a secure way: the sender's public key is not provided by the sender, but is provided by a trusted external entity through a public key certificate, otherwise an attacker could pass his own public key off as the legitimate sender's public key.

Usually a certificate binds a public key to an identity, that is a natural person, but attributes depend on the application context:

- network security: attributes are IP addresses;
- e-mail security: attributes are e-mail addresses;
- web security: attributes are URLs.

Formats for public key certificates

- PKCS #6: the old RSA standard, obsoleted by X.509 version 3;
- X.509 (sect. 3.2): it is the currently most used standard:

  - versions 1 (1988) and 2 (1993): they were defined only by ISO as an integral part of the OSI system, but they are not very satisfactory;
  - version 3 (1996): it was defined by ISO jointly with IETF, and is suitable for internet security needs thanks to the introduction of extensions and certificate attributes;
- others (e.g. PGP[1], SPKI): alternative solutions, created by individuals who refuse the usage of X.509, which however have almost no practical usage.

3.1.1 PKI

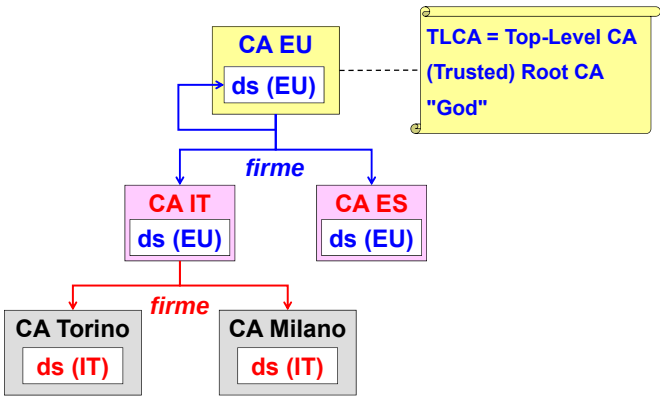


Figure 3.2: Certification hierarchy.

public-key infrastructure (PKI) the technical and administrative infrastructure put in place for the creation, distribution and revocation of public key certificates

certification authority (CA) an entity that issues digital certificates (especially X.509 certificates) and vouches for the binding between the data items in a certificate

registration authority (RA) an optional PKI entity (separate from the CAs) that does not sign either digital certificates or CRLs but has responsibility for recording or verifying some or all of the information (particularly the identities of subjects) needed by a CA to issue certificates and CRLs and to perform other certificate management functions

end entity a system entity that is the subject of a public-key certificate and that is using, or is permitted and able to use, the matching private key only for purposes other than signing a digital certificate; i.e., an entity that is not a CA

relying party a system entity that depends on the validity of information (such as another entity's public key value) provided by a digital certificate

certification hierarchy a tree-structured (loop-free) topology of relationships between CAs and the entities to whom the CAs issue public-key certificates

root CA the CA that is the highest level (most trusted) CA in a certification hierarchy; i.e., the authority upon whose public key all certificate users base their validation of certificates, CRLs, certification paths, and other constructs

self-signed certificate a public-key certificate for which the public key bound by the certificate and the private key used to sign the certificate are components of the same key pair, which belongs to the signer

Public-Key Infrastructure (PKI) is made up of entities such as:

[1]Please refer to section 8.7.1.

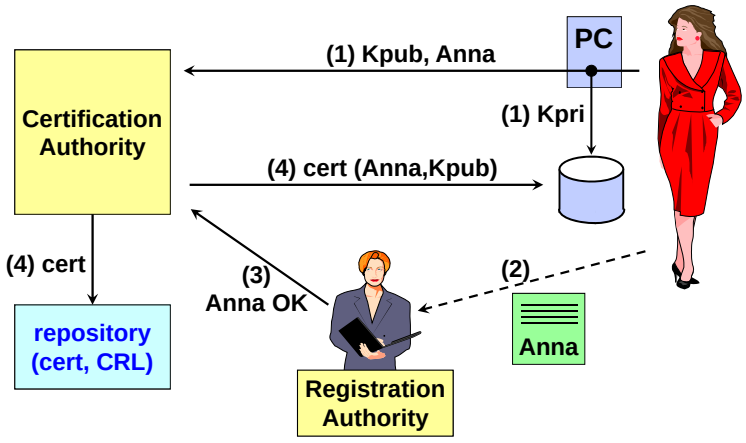
- **Certification Authority** (CA): it is the technical component and is in charge of issuing public key certificates;
- **Registration Authority** (RA): it is the administrative component and is in charge of verifying the identity of who is requesting a certificate.

The key is bound in a secure way thanks to the fact that the certificate is electronically signed by the issuer, that is the CA, which is a trusted external entity ⇒ the digital signature of the CA attached to the certificate provides integrity and authentication, preventing from creating fake certificates. By this system, the certificate which is used to verify a digital signature contains the digital signature of the CA, which would have to be verified by another certificate in turn containing the digital signature of another CA, and so on endlessly.

CAs are organized in a **certification hierarchy**: the digital signatures contained in the certificates of a CA are verified by certificates of the CA at the upper level. At the top of the certification hierarchy there is a **top-level CA**, which issues **self-signed certificates**, that is signed by itself. Only the self-signed certificates of the CAs included in a list of trusted top-level CAs are considered as valid; this list is stored in the user's device and preconfigured by the vendor (other top-level CAs can be added by the user).

TSL The **Trust service Status List** (TSL) is a signed list published by the European Union to communicate the Trust-Service Providers (TSP), that is the trusted authorities (e.g. CAs, OCSP servers, Time-Stamping Authorities), for each member state.

3.1.2 Certificate issuance



issue generate and sign a digital certificate (or a CRL) and, usually, distribute it and make it available to potential certificate users (or CRL users)

subject the name (of a system entity) that is bound to the data items in a digital certificate; e.g., a DN that is bound to a key in a public-key certificate

issuer the CA that signs a digital certificate or CRL

repository a system for storing and distributing digital certificates and related information (including CRLs, CPSs, and certificate policies) to certificate users

In general a certificate can also be created directly by the user, but an attacker could claim to own the private key corresponding to someone else's public key ⇒ the CA has to certify the binding between the public key and the identity of the owner of the corresponding private key.

A public key certificate is generated in the following way:

1. the user generates the key pair which constitutes his own asymmetric key:
   - the private key is locally stored on the user's device, and the task of preserving it so that it will keep being secret is left to the user;
   - the public key is sent to the CA through the network, together with the user's identity (e.g. by a PKCS #10 request: please refer to section 3.9);
2. the user goes in person to a RA and delivers a document attesting his identity;
3. the RA notifies to the CA that the request for a certificate can be accepted;
4. the CA issues and sends to the user a public key certificate containing the binding between the user's identity and the public key;
5. the CA publishes the new certificate into a public **repository**, from which all the certificates issued by the CA (including revocation information, e.g. CRL) can be retrieved.

3.1.3 Certificate revocation

> **certificate revocation** the event that occurs when a CA declares that a previously valid digital certificate issued by that CA has become invalid; usually stated with an effective date

The certificate has a limited lifetime, and hence has to be periodically renewed. It can be revoked before its natural expiration:

- upon a request by the owner of the public key (**subject**): he can ask for the certificate revocation if he has no longer control of his own private key (e.g. it has been stolen);
- autonomously by the CA (**issuer**): it can ask for the certificate revocation if it discovers that it has been defrauded (e.g. the user claimed to be someone else).

When validating a digital signature, the task of checking if the certificate was valid or had been revoked at signing time is left to the receiver (**relying party**), through two possible mechanisms:

- CRL (sect. 3.3): the user downloads the list, signed by the CA or a delegate (e.g. RA), containing all revoked certificates with their revocation dates:
  - \+ delayed verification: the validity of a certificate can be checked, even off-line, at any past date (e.g. at signing time);
  - − big reply: a big list needs to be downloaded;
- OCSP (sect. 3.4): the user queries a server to get the status for a single certificate:
  - \+ real-time verification: the validity refers to the query time;
  - − small reply: the server just returns 'good', 'revoked' o 'unknown'.

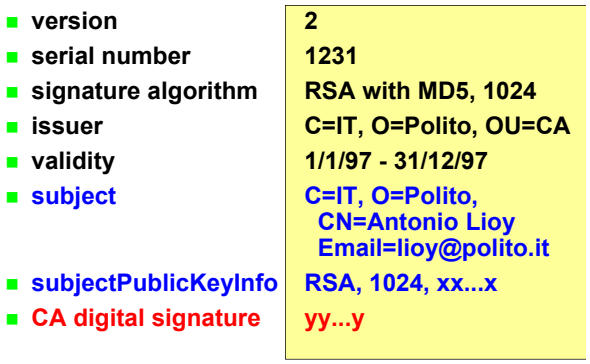
3.2 X.509 public key certificates

3.2.1 Structure of a X.509 certificate

The X.509 standard is described by using the **Abstract Syntax Notation 1** (ASN.1) syntax, a notation which allows to specify any data format in a completely abstract way.

A X.509 version 3 certificate contains, besides the CA's digital signature, a sequence of 10 data items, including:

1. `version`: it is the certificate encoding version (= X.509 standard version −1);



2. `serialNumber`: it is an integer assigned by the CA, unique for each certificate issued by a given CA (i.e. the issuer name and serial number identify a unique certificate);
3. `signature`: it contains the algorithm identifier for the algorithm and hash function used by the CA in signing the certificate (e.g. `sha1WithRSAEncryption`);
4. `issuer`: it is the **distinguished name** (DN) which identifies the entity that has signed and issued the certificate (ad es. `countryName=IT`, `stateOrProvinceName=Turin`, `organizationName=Politecnico di Torino`, `commonName=Test CA`);
5. `validity`: it is the time inverval, comprised between `notBefore` and `notAfter`, where the private key corresponding to the public key being certified;
6. `subject`: it is the DN which identifies the entity associated with the public key being certified (e.g. `commonName=Antonio Lioy`);
7. `subjectPublicKeyInfo`: it carries the value in clear of the public key being certified, and identifies the algorithm which this public key is an instance of (e.g. `rsaEncryption`: public exponent and modulus);
10. `extensions` (optional): it contains possible certificate-specific extensions.

3.2.2 Extensions

Extensions were introduced by version 3 of X.509 to extend certificate and CRL definitions, making them more flexible and accommodating different needs.

Marking an extension as critical or non-critical is the responsibility of the CA:

- non-critical: if the receiver does not recognize the extension, he can choose whether reject the certificate or skip the extension;
- critical: if the receiver does not recognize the extension, he shall reject the whole certificate. Processing is entirely the responsibility of the relying party: he may still decide to accept the certificate at his own risk, without any legal coverage by the CA.

There are two types of extensions:

- standard: extensions defined by the standard, hence understood by all receivers, split into five categories:
  - **key and policy information**: they provide information about the key being certified and the security policy, that is the formal rules followed by the CA when issuing the certificate;
  - **subject and issuer attributes**: they provide additional information about the subject and the issuer;

  - **certification path constraints**: they enforce constraints on the certification chain;
  - **basic CRL extensions**: they provide basic information about the CRL;
  - **CRL distribution points and delta-CRLs**: they provide information about CRL distribution points and delta CRLs;

- private: extensions defined by a single company or closed group, hence understood only by a restricted group of receivers (e.g. within the organization).

Standard extensions

Certificate-specific standard extensions include:

- key and policy information:

  a) **authority key identifier** extension: it identifies the public key to be used to verify the signature on this certificate;

  c) **key usage** extension: it identifies the range of cryptographic applications, that is the purposes for which the public key can be used (e.g. digital signature, key agreement);

  d) **extended key usage** extension: it identifies the range of end-user applications, which can use multiple cryptographic applications (e.g. code signing, server authentication);

- subject and issuer attributes:

  1. **subject alternative name** extension: it binds to the subject of the certificate one or more unique names (e.g. e-mail address, IP address);
  2. **issuer alternative name** extension: it binds to the issuer of the certificate one or more unique names (e.g. e-mail address, IP address);

- certification path constraints:

  a) **basic constraints** extension: it specifies whether the subject is a CA or a user (**end entity**) in the certification tree, and in the former case the maximum depth of the subtree can be specified;

  b) **name constraints** extension: it restricts the domain of trustworthy names which the subject CA can insert into its certificates (e.g. e-mail addresses restricted to "*.polito.it");

- CRL distribution points and delta-CRLs:

  a) **CRL distribution points** extension: it contains the directory entries, e-mail addresses or URLs of one or more distribution points from which CRLs to be used for verifying the validity of a certificate can be retrieved.

Private extensions

The IETF's PKIX group has defined in an RFC three private extensions which, given their importance, have become practically standard:

- **subject information access** extension: it specifies a method (e.g. http) to retrieve information about the owner of a certificate and a name to indicate where to locate it (address);
- **authority information access** extension: it is a back-pointer to the services of the CA that issued a certificate (e.g. URL of the OCSP server[2]);
- **CA information access** extension: it is a back-pointer to the services of the CA owning the certificate.

[2]Please refer to section 3.4.

3.3 CRL X.509

certificate revocation list (CRL) a data structure that enumerates digital certificates that have been invalidated by their issuer prior to when they were scheduled to expire

delta CRL a partial CRL that only contains entries for certificates that have been revoked since the issuance of a prior, base CRL

indirect certificate revocation list (ICRL) in X.509, a CRL that may contain certificate revocation notifications for certificates issued by CAs other than the issuer (i.e., signer) of the ICRL

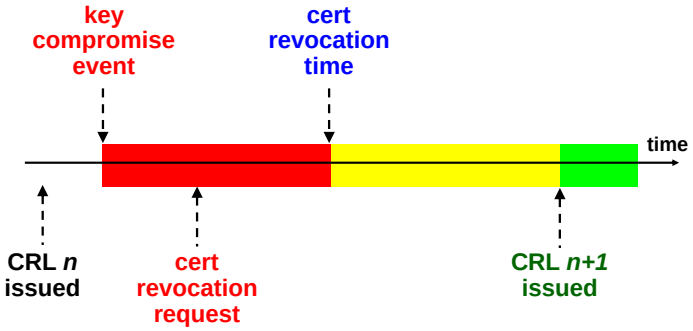
A **Certificate Revocation List** (CRL) can be:

- base CRL: contains the list of all the certificates that have been revoked before a certain date, with their revocation dates;
- delta CRL: contains the list of just the revoked certificates since a previous base CRL was issued, to provide smaller, more dynamic **differential updates**.

CRLs are periodically issued by certificate issuers to keep them updated.
CRLs can be signed by two different entities:

- directly by the CA that issued the revoked certificates;
- by a RA delegated by the CA (**indirect CRL** [iCRL]).
  A CA can delegate a RA for organizational reasons: the RA can accept revocation requests and issue a new CRL even outside the CA employees' working hours.

3.3.1 Certificate revocation timeline



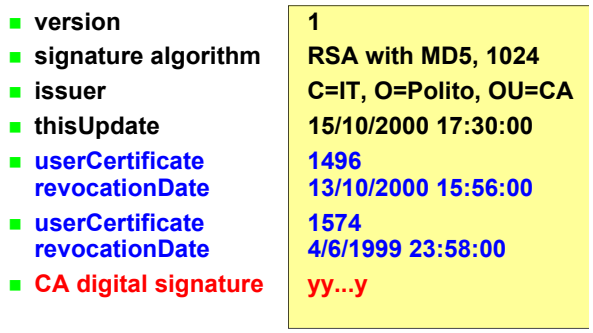
The certificate revocation is not immediate, but some time, during which the signatures made by the stolen key are considered as valid, elapses from the compromise event to the updated CRL issuance:

1. CRL issuance: CRL number n is issued;
2. key compromise event: the user's private key is stolen or lost;
3. certificate revocation request: the user goes to the CA to ask for the revocation of the certificate bound to the stolen key, claiming the moment in which the compromise event occurred.
   This is the **invalidity date**: the owner of the public key, not the CA, has the responsibility for its truthfulness;
4. certificate revocation time: the CA, after checking that the requester's identity is corresponding to the certificate subject's identity, revokes the certificate.
   This is the **revocation date**: the CA has the responsibility for its truthfulness;
5. CRL issuance: CRL number $n+1$, where the certificate revocation is made public, is issued.

Attacks The CRL is practically useless if it is not up-to-date or if when the signature was made is not exactly known:

- back-dating attack: the attacker makes the signature bringing back the date on the device, so that the signature will appear to be made before the key theft ⇒ a proof is needed that the signature was made before a certain time instant (please refer to section 3.5);
- DNS spoofing attack: the 'issuing distribution point' field in the CRL is not protected ⇒ an attacker can redirect the relying party to an old version of the CRL.

3.3.2 Structure of a X.509 CRL



A X.509 version 2 CRL contains, besides the issuer's digital signature, a sequence of 7 data items:

1. `version`: it is the CRL encoding version;[3]
2. `signature`: it contains the algorithm identifier for the algorithm and hash function used by the issuer in signing the CRL (e.g. `sha1WithRSAEncryption`);
3. `issuer`: it is the DN which identifies the entity that has signed and issued the CRL;
4. `thisUpdate`: it is the date and time at which the CRL was issued;
5. `nextUpdate` (optional): it is the date and time by which the next CRL will be issued;
6. `revokedCertificates` (optional): it lists the certificates that have been revoked:
   - `serialNumber`: it is the serial number of the revoked certificate;
   - `revocationDate`: it is the revocation date and time of the certificate, that is when the owner of the public key performed the revocation request to the CA;
   - `crlEntryExtensions` (optional): it contains possible entry (revoked certificate)-specific extensions;
7. `crlExtensions` (optional): it contains possible CRL-specific extensions.

3.3.3 Extensions

CRL-specific extensions

CRL-specific standard extensions (`crlExtensions`) include:

- key and policy information:
  a) **authority key identifier** extension: it identifies the public key to be used to verify the signature on this CRL;

[3]The v1 and v2 designations for an X.509 CRL are disjoint from the v1 and v2 designations for an X.509 public-key certificate, and from the v1 designation for an X.509 attribute certificate.

- subject and issuer attributes:

  b) **issuer alternative name** extension: it binds to the issuer of the CRL one or more unique names (e.g. e-mail address, IP address);

- basic CRL extensions:

  a) **CRL number** extension: it is the progressive sequence number of the CRL;

- CRL distribution points and delta-CRLs:

  b) **issuing distribution point** extension: it contains the directory entry, e-mail address or URL of the distribution point from which the next CRL can be retrieved;

  e) **delta CRL indicator** extension: it identify the CRL as being a delta CRL providing differential updates to a referred base CRL.

Entry-specific extensions

Entry-specific standard extensions (`crlEntryExtensions`) include:

- basic CRL extensions:

  b) **reason code** extension: it specifies the reason for which the certificate was revoked (some reasons are more severe than other ones);

  c) **hold instruction code** extension (deprecated): it allows to temporarily pause the usage of a certificate (e.g. a soldier goes on vacation);

  d) **invalidity date** extension: it is the date and time since which the certificate is no longer valid, that is when the key compromise event occurred;

- CRL distribution points and delta-CRLs:

  d) **certificate issuer** extension: when the CRL is indirect, it identifies the CA which issued the revoked certificate and delegated the RA.

3.4 OCSP

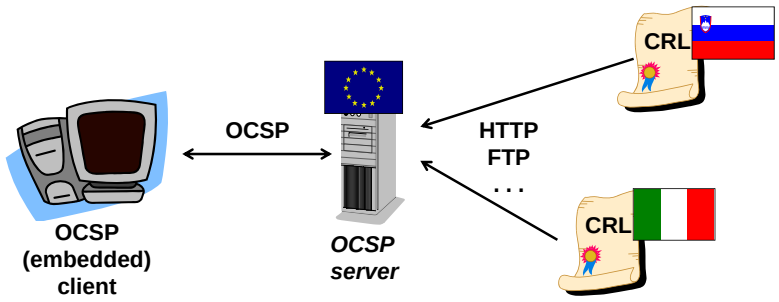


Figure 3.3: OCSP architecture.

certificate status responder a trusted online server that acts for a CA to provide authenticated certificate status information to certificate users; offers an alternative to issuing a CRL

The **Online Certificate Status Protocol** (OCSP) standard, defined by the IETF-PKIX group, is used to check on-line whether a certificate is valid:

1. an **OCSP server** (or OCSP responder) downloads all the CRLs from which to obtain revocation information;
2. the relying party queries the OCSP server on the current status of the certificate;
3. the OCSP server returns the current status of the certificate:
   - good: the certificate has not been revoked;
   - revoked: the certificate has been revoked:
     - `revocationTime`: the certificate revocation date and time;
     - `revocationReason`: the certificate revocation reason;
   - unknown: the certificate is unknown.

The responses are signed by the OCSP server, but the OCSP server certificate can not be verified by OCSP itself ⇒ often OCSP server certificates have a very short lifetime (e.g. 24 hours).

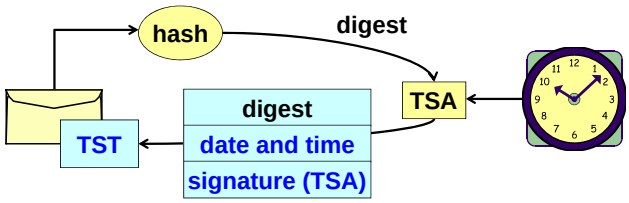
OCSP server models

- **Trusted Responder**: the OCSP server is paid by users:
  - it answers requests for any certificates;
  - the OCSP server certificate is independent of CAs;
  - users trust the server (e.g. corporate server, trusted third party);
- **Delegated Responder**: the OCSP server is paid by a CA:
  - it only answers the requests for the certificates issued by the CA;
  - the OCSP server certificate is provided by the CA ('extended key usage' extension = `ocspSigning`);
  - users retrieve the OCSP server URL from the 'authority information access' extension in the certificate to be verified.

DoS attacks Signing the response causes a big load on the server itself ⇒ responses can be pre-computed to decrease the server load, but:

- they are no longer updated in real time;
- they make possible replay attacks.

3.5 Time-stamping



time stamp with respect to a data object, a label or marking in which is recorded the time (time of day or other instant of elapsed time) at which the label or marking was affixed to the data object

The **Time-Stamp Token** (TST) is a proof attached to a document in order to attest that:

- the document was created before a certain time instant (the exact creation time is not known);
- the document has not been changed after that time instant.

The TST is provided by a trusted third party, called **Time-Stamping Authority** (TSA), through a client-server protocol, called **Time-Stamp Protocol** (TSP):

1. the client sends to the TSA the document digest computed through a hash algorithm (the document could be confidential);
2. the TSA returns a TST containing:
   - the document digest;
   - the current date and time provided by the TSA clock;
   - the TSA's digital signature, computed on the date and time and the digest;
3. the client attaches the TST to the document.

This system is useful at application level when a document has to be completed within a certain deadline, but its delivery could be delayed (e.g. the server on which to upload it is overloaded).

A single TST is not enough to distinguish the creation time and the signing time of a document ⇒ by two TSTs it is possible to prove that the document was signed in the time interval comprised between the first and the second TST:

1. a first TST is attached to the document without signature;
2. the document is signed by the user by its own private key;
3. a second TST, also covering the signature, is attached to the signed document.

3.6 PSE

personal security environment (PSE) secure local storage for an entity's private key, the directly trusted CA key and possibly other data; depending on the security policy of the entity or the system requirements, this may be, for example, a cryptographically protected file or a tamper resistant hardware token

The **Personal Security Environment** is a place where a user can keep:

- his own private key: it must be kept secret;
- self-signed certificates of trusted root CAs: they must be authentic (not fake).

The PSE can be implemented in two ways:

- **software** (sect. 3.8): the private key and certificates are saved inside an encrypted file;
- **hardware** (sect. 3.7): the private key and certificates are saved inside a hardware device.

A PSE supports **mobility**: the user can use his own key on several devices, even though issues can arise:

- hardware: a device (e.g. smartphone) could not have a USB port to which to connect the smart card reader;
- software: the application could not support the PKCS #12 format.

3.6.1 PKCS #11

PKCS #11 is a low-level security API which offers to application developers a standard interface to access primitives in the lower-level PSE.

It provides a **cryptographic engine** able to perform basic cryptographic operations (e.g. signing, encryption), which can be implemented:

- in software: a library implementing all algorithms;
- in hardware: a library which is a stub toward a hardware PSE.

Crypto Service Provider (MS-CAPI CSP) is Microsoft's proprietary alternative.

3.6.2 Secure binary data formats

A better alternative to using cryptographic primitives is represented by secure **binary** data **formats**:

- PKCS #12 (sect. 3.8): format implementing software PSEs for storing and transporting private keys and certificates;
- PKCS #10 (sect. 3.9): format for certificate requests in a certification infrastructure;
- PKCS #7 (sect. 3.10): format for secure envelopes = data structure where to put the data to be signed and/or encrypted.

Application formats, such as the S/MIME standard[4] and standards for creating legal electronic documents, are based on these binary formats.

3.7 Hardware PSEs

A hardware PSE can be:

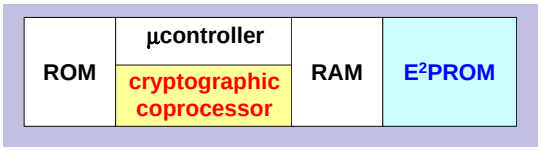
- **passive** (e.g. USB flash drive): it just stores the private key and certificates in an encrypted form, e.g. digital signature:
  1. the device provides the private key as the input to the PC;
- **active**: the device is able to internally perform cryptographic operations thanks to on-board integrated cryptographic functions, e.g. digital signature:
  1. the PC provides the digest as the input to the device;
  2. the device returns the digital signature as the output to the PC.

Secret keys can be generated:

- externally: the key needs to be externally generated and then inserted into the device ⇒ in case of device fault or loss, the encrypted data can still be decrypted if the key has been backed up in a safe place;
- internally: the device is able to compute the key autonomously ⇒ the key always stays confined within a protected environment (the PC could be infected by viruses).

[4] Please refer to section 8.7.5.

3.7.1 Cryptographic smart card



cryptographic token a portable, user-controlled, physical device (e.g., smart card or PCMCIA card) used to store cryptographic information and possibly also perform cryptographic functions

cryptographic smart card chip card with memory and autonomous cryptographic capability

A cryptographic smart card includes:

- microcontroller: a simple processor, with integrated I/O systems, which having a limited computational power is not suitable to perform cryptographic operations;
- ROM: it stores the smart card operating system (CardOS);
- **cryptographic coprocessor**: hardware circuits implementing cryptographic functions (e.g. signing);
- **EEPROM**: a sort of non-volatile flash memory, of small size (4-32 KB), where secret keys are stored in an encrypted form.

A variety of smart cards exists:

- simple (cheap): the cryptographic coprocessor implements a symmetric algorithm (DES);
- complex (expensive): the cryptographic coprocessor implements an asymmetric (RSA) or elliptic curve algorithm.
  The cost depends on:
  - key length;
  - on-board generation of private keys.

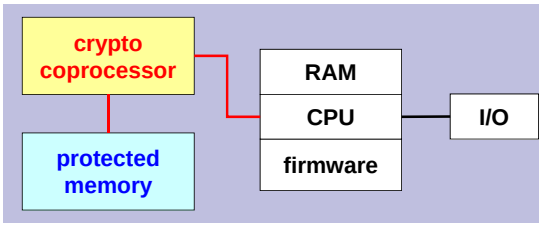
Disadvantages

- using smart cards requires a **smart card reader**:
  - drivers for all existing operating systems are needed;
  - support by applications is needed.
- smart cards are for users, not for servers:
  - I/O interfaces allow slow transfers;
  - cryptographic operations are slow.

  An attacker could send to an active smart card a lot of digest to be signed in order to make the smart card busy ⇒ the user can be asked to type a **pin**, by means of an on-board keypad, whenever a signature is generated.

3.7.2 HSM

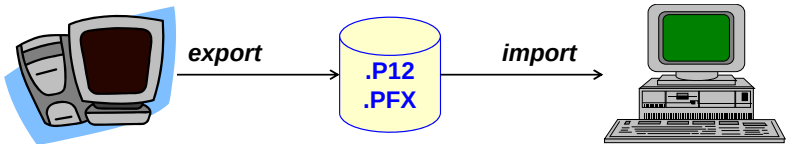
The **Hardware Security Module** (HSM) is a cryptographic accelerator for servers (e.g. OCSP server, Time-Stamping Authority) which has the same logical components as the smart card, but with better performance.



Form factors

- PCI cards;
- external devices: USB, IP, SCSI...

3.8 PKCS #12



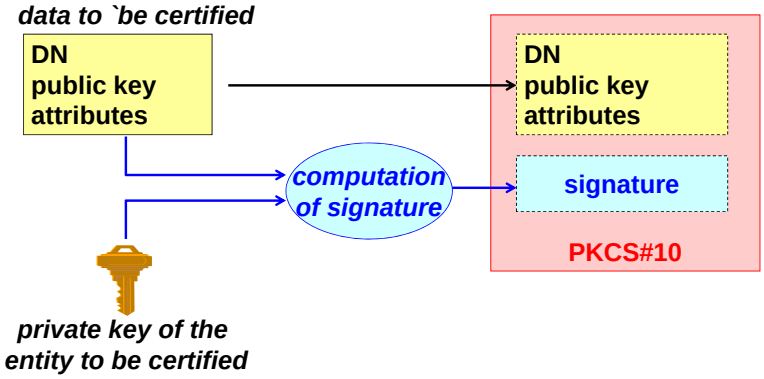
PKCS #12 is the standard format, derived from the Microsoft PFX format, which implements a software PSE:

- transport: the digital identity can be exported and imported among different applications (e.g. web browsers) or systems;
- backup: the private key can be recovered in case of loss.

A PKCS #12 file contains personal cryptographic material, signed and encrypted by password, such as:

- the private key;
- the certificate corresponding to the private key;
- the issuing CA's certificate.

3.9 PKCS #10



PKCS #10 is the standard format used by a user for a certificate request to the CA.
The PKCS #10 request includes:

- the requester user's distinguished name (DN);
- the public key to be certified;
- possible attributes, including:
  - a challenge password by which the user may later request certificate revocation;
  - the attributes (e.g. PKCS #9) which have to be inserted in the certificate;
  - other information about the requester.

proof-of-possession a protocol whereby a system entity proves to another that it possesses and controls a cryptographic key or other secret information

The PKCS #10 request is signed by the private key corresponding to the public key to be certified ⇒ the CA can check whether the request is the legitimate owner of the public key.

3.10 PKCS #7

PKCS #7, named 'Cryptographic Message Syntax' (CMS), allows to create **secure envelopes** which can contain data of any kind.

PKCS #7 is very flexible:

- data can be encrypted and/or signed by using either symmetric or asymmetric algorithms;
- it allows multiple signatures (parallel or sequential) over a same object (please refer to section 3.11.2);
- it is a **self-consistent format**: it can include, besides the data and the signature, also the certificates (with their revocation information) needed to verify the signature;
- it is a **recursive format**: the content of a PKCS #7 object can be another PKCS #7 object.

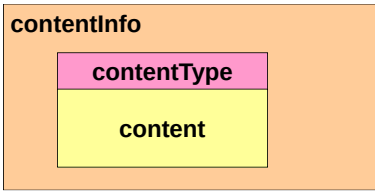


Figure 3.4: PKCS #7 structure.

A PKCS #7 envelope contains a high-level block (`contentInfo`) split into two parts:

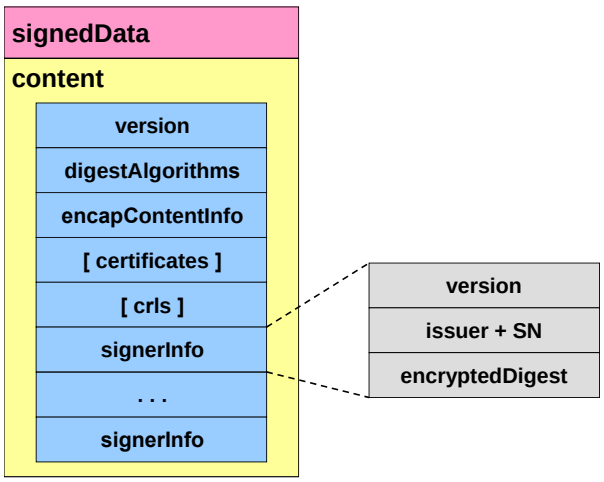
- `contentType`: the type of basic content;
- `content`: the data + the additional information about the data.

The PKCS #7 standard defines six types of basic content:

- `data`: data in clear (encoding of a generic sequence of bytes);
- `signedData` (sect. 3.10.1): data in clear + one or more parallel signatures;
- `envelopedData` (sect. 3.10.2): data encrypted by a symmetric key + symmetric key encrypted by the recipients' RSA public keys;

- `signedAndEnvelopedData`: data + digital signatures, both encrypted by the recipients' RSA public keys;
- `digestData`: data in clear + digest (for integrity only);
- `encryptedData`: data encrypted by a symmetric key (not included because assumed to be managed by other means).

3.10.1 Digitally-signed documents (enveloping signature)



The `signedData` type is the most used one to envelop signed data:

- `version`: the used standard version;
- `digestAlgorithms`: the hash algorithms used to create the signatures;
- `encapContentInfo`: the structure containing the encapsulated data (in clear);
- `certificates` (optional): the list of the certificates needed to verify the signatures;
- `crls` (facoltativo): the list of the CRLs needed to verify the signatures;
- `signerInfo`: the information about one of the signers:
  - `version`: the version number;
  - `issuer + SN`: the issuer's name and the certificate serial number which uniquely identify the signer's certificate;
  - `encryptedDigest`: the digital signature.

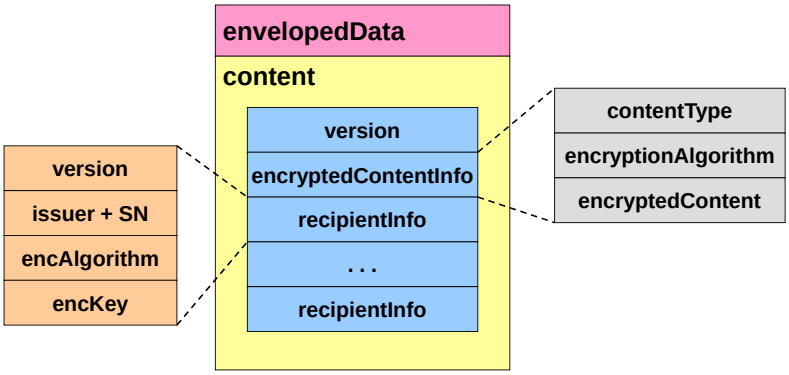
The `digestAlgorithms` field is put at the beginning of the envelope ⇒ the receiver, while reading the encapsulated data, can immediately start computing digests on them, in order to be able to later compare them with the ones included in the envelope quickly.

3.10.2 Digital envelopes

digital envelope a combination of

(a) encrypted content data (of any kind) intended for a recipient

(b) the content encryption key in an encrypted form that has been prepared for the use by the recipient

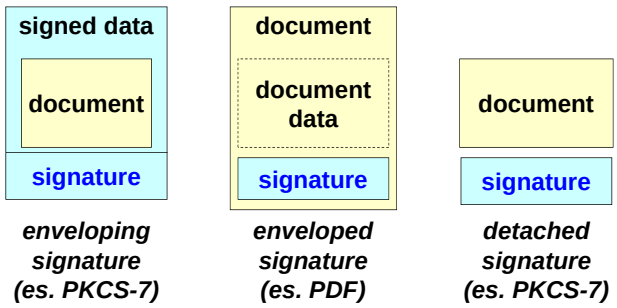


The `envelopedData` type is the most used one to envelop encrypted data:

- `version`: the used standard version;
- `encryptedContentInfo`: the structure including the encapsulated encrypted data:
  - `contentType`: the content type;
  - `encryptionAlgorithm`: the symmetric cryptographic algorithm used to encrypt the data;
  - `encryptedContent`: the actual encapsulated data, encrypted by the symmetric key;
- `recipientInfo`: the information about one of the recipients (= receivers authorized to read the content):
  - `version`: the version number;
  - `issuer + SN`: the issuer's name and the certificate serial number which uniquely identify the recipient's certificate containing the public key used to encrypt the symmetric key;
  - `encAlgorithm`: the asymmetric cryptographic algorithm used to encrypt the symmetric key;
  - `encKey`: the symmetric key used to encrypt the data, encrypted by the recipient's public key.

3.11 Digitally-signed documents

3.11.1 Formats of signed documents



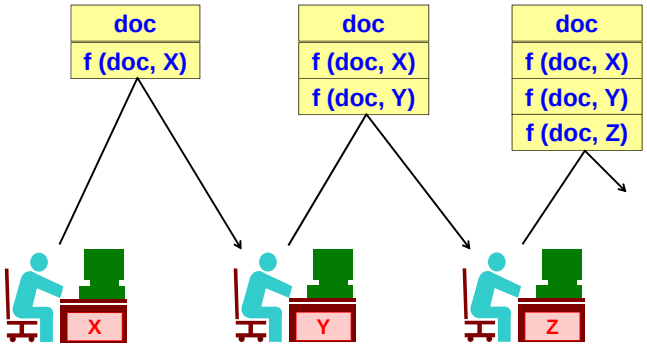
At a conceptual level, the digital signature can be attached to the document in three formats:

- **enveloping signature** (e.g. PKCS #7: sect. 3.10.1): the signature wraps the document ⇒ the document should be extracted before being able to be read;

- **enveloped signature** (e.g. PDF): the document wraps the signature, that is the document format contemplates a blank for the signature;
- **detached signature** (e.g. PKCS #7): the document and the signature are detached from each other, and the signature includes a reference (e.g. filename) to the document ⇒ problem: how to keep over time the binding between the signature and the document?

3.11.2 Multiple signatures

Parallel signatures

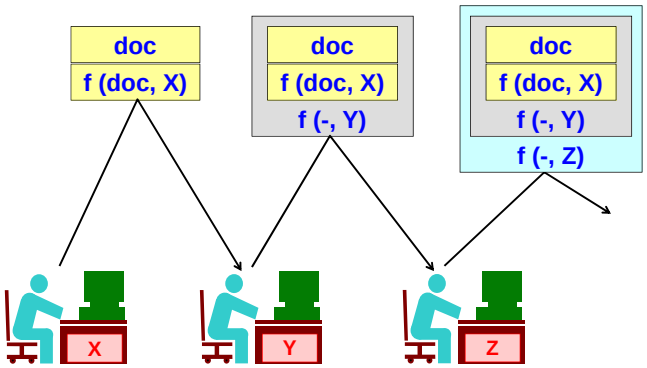


A single PKCS #7 envelope can include multiple **parallel** (or independent) **signatures**:

1. user X creates the document and signs it;
2. user Y adds his own signature related to the document;
3. user Z adds his own signature related to the document, and so on.

Disadvantage The order in which the signatures have been attached can not be known.

Sequential signatures



Multiple PKCS #7 envelopes can be recursively included one inside the other, each one including one of the **sequential** (or hierarchical) **signatures**:

1. user X creates the document and signs it;
2. user Y adds his own signature related to the document including user X's signature;
3. user Z adds his own signature related to the document including the previous signatures, and so on.

Advantage The order in which the signatures have been attacked can be known, because signers are bound together by a hierarchy (e.g. researcher → department boss → university rector).

3.11.3 Legal value

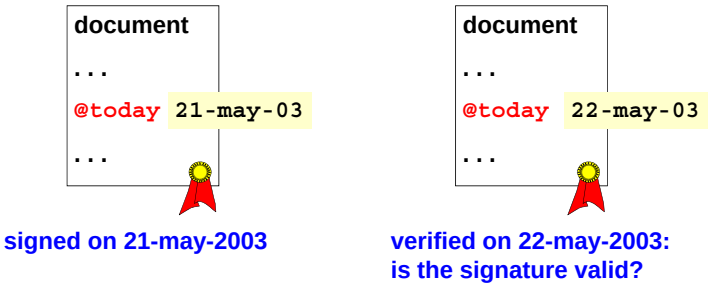


Figure 3.5: The signature is valid, despite the presence of the macro: not the bits, but just the final view of the document is changed.

A digital signature offers the non-repudiation property, but a lot of aspects need to be considered by a court of justice so that its **legal value** is recognized:

- syntax: it this your signature? (all bits should match)
- semantics: did you understand what you were signing? (e.g. white text over white background does not follow WYSIWYS = 'What You See Is What You Sign')
- will: did you sign voluntarily? (a notary should check the contractor's will)
- identification: was really you the signer? (e.g. stolen smart card; a notary should identify the contractor)
- time: when did you sign? (the date can be manipulated)
- place: where did you sign? (if the electronic document has some tax value, different countries can have different taxations)

3.12 European electronic signature

> **electronic signature** the set of data in an electronic form which is attached or logically bound to other data in an electronic form and provides its means of authentication

The **electronic signature** (ES) is a signature which has legal value at level of European Union.

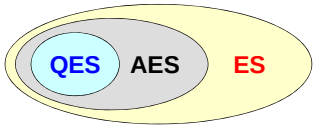
The ES is a general concept which can include various kinds of signatures with different uncertainty levels:

- a digital signature;
- a scanned handwritten signature;
- the code returned by a special pen recognizing hand movements.

Member states shall ensure that an ES is not denied legal value just because:

- it is in an electronic form;
- it is not based upon Qualified Certificates;

- it does not use certificates issued by accredited certifiers;
- it was not created by a secure signature device.



Depending on the uncertainty level:

- AES (sect. 3.12.1): its legal value should be discussed by a court of justice;
- QES (sect. 3.12.2: it automatically has legal value equivalent to the handwritten signature, thanks to its minimum uncertainty level.

3.12.1 AES

An **Advanced Electronic Signature** (AES) is an ES which meets the following requirements:

- it is in a unique relation with the signer (just he might have made it);
- it allows to identify the signer;
- it was created by using tools which the signer can keep under his control (e.g. smart card);
- it is in relation with the data which it is referring to so that any further data modification will be detectable (e.g. digest).

These requirements are not bound to a certain technology, but they are defined independently of technological evolution in order to keep being valid in the future.

3.12.2 QES

A **Qualified Electronic Signature** (QES) is an AES which meets the following requirements:

- it was attached by using a Qualified Certificate;
- it was attached by using a signature device certified as being secure.

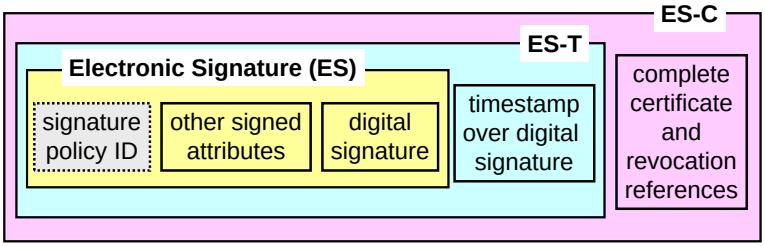
> **qualified certificate** a public-key certificate that has the primary purpose of identifying a person with a high level of assurance, where the certificate meets some qualification requirements defined by an applicable legal framework, such as the European Directive on Electronic Signature

A **Qualified Certificate** (QC) is a certificate which ensures a person's identity and includes:

- the indication that it is a QC;
- the indication of the certifier and the status in which it was issued;
- indications about usage limitations for the certificate;
- indications about the limit for commercial transactions which can be made by that certificate.

3.12.3 CAdES

The **CMS Advanced Electronic Signatures** (CAdES) is the European standard format for electronic signatures, published by the ETSI entity and based on the CMS (PKCS #7) format.



Types of CAdES formats

- ES: it contemplates:
  - digital signature;
  - other signed attributes;
  - signature policy identifier;
- ES-T (timestamp): a timestamp is computed upon the signature to prevent back-dating attacks;
- ES-C (complete): it includes, besides the signature with timestamp, also the references to all the certificates and revocation information needed to check the signature validity;
- ES-X (extended): they offer a further security when CAs' certificates may be compromised:
  - ES-X-Timestamp (type 1) (with OCSP): the timestamp is applied to the whole ES-C;
  - ES-X-Timestamp (type 2) (with CRL): the timestamp is only applied to certificate and revocation information references.

Application signature formats CAdES is a binary signature format which some application signature formats are derived from, such as:

- XML Advanced Electronic Signatures (XAdES): based on XML-dsig, digital signature standard for XML documents;
- PDF Advanced Electronic Signature Profiles (PAdES): electronic signature standard in PDF documents.

Chapter 4

Authentication systems

4.1 Authentication methodologies

Authentication methodologies can be based on different mechanisms:

1. 'something the user knows' (e.g. password, PIN);
2. 'something the user owns' (e.g. magnetic card);
3. 'something the user physically is' (biometrics, e.g. fingerprint).

Different mechanisms can be combined to achieve a higher and higher level of authentication, up to the real identification[1]:

- **one-factor authentication**: first mechanism;
- **two-factor authentication**: first mechanism + second mechanism (e.g. ATM);
- **three-factor authentication**: first mechanism + second mechanism + third mechanism.

With the diffusion of mobile phones another mechanism could be added, which is the user position, always known by the cellular network.

4.1.1 One-factor authentication

authentication the process of verifying a claim that a system entity or system resource has a certain attribute value

identification an act or process that presents an identifier to a system so that the system can recognize a system entity and distinguish it from other entities

verification the process of examining information to establish the truth of a claimed fact or value

The system is made up of:

- user: it is identified by a User ID (UID), and has knowledge of a secret S_{UID} (e.g. password);
- server: it contains a table with $\{\text{UID} : f(S_{\text{UID}})\}$ pairs, where f is a function computed on the secret.

[1] Authentication is a different concept from **identification**:

- authentication: it is the result of an electronic procedure (e.g. entering username and password);
- identification: it means to be sure that the login credentials have really been entered by their holder and not by third persons.

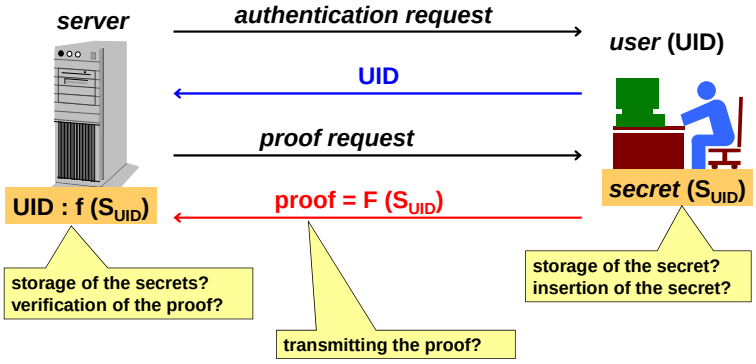


Figure 4.1: Reference scheme for an authentication system based on the first mechanism.

The authentication process acts before the communication starts:

1. **identification**: presenting the claimed attribute value to the authentication subsystem:
   (a) authentication request: the server asks the user to authenticate himself;
   (b) UID: the user communicates his own UID to the server (e.g. username);
2. **verification**: presenting or generating authentication information that acts as evidence to prove the binding between the attribute and that for which it is claimed:
   (a) proof request: the server asks the user for a proof, which proves that he really is who is claiming to be;
   (b) proof: the user provides as a proof the result from function F applied to secret S_{UID}: proof $= F(S_{\text{UID}})$;
   (c) check: the server compares the received proof with stored secret S_{UID}: if the proof is the expected one, the server gives the user access to the services it offers.

Issues

- client side:
  - how should the user insert the secret to be sent to the server?
  - how can the user's secret be kept safe?
- network side: the proof could be cancelled or manipulated at will by a man-in-the-middle: how can it be transmitted in a secure way?
- server side:
  - how can the server verify the proof?
  - the server needs to know the user's secret: how can it be securely stored in the server?

4.2 Storing passwords on the server

How to save passwords on the server in a secure way?

- in clear: all passwords are stored in clear (f = identity function I):

$$f(S_{\text{UID}}) = P_{\text{UID}}$$

  – anyone having access to the server is able to read all users' passwords;

- encrypted: all passwords are stored encrypted (f = encryption function enc):

$$f(S_{\text{UID}}) = \text{enc}(K, P_{\text{UID}})$$

  - − the decryption key should be saved in clear;

- hashed: only password digests are stored (f = hash function h):

$$f(S_{\text{UID}}) = h(P_{\text{UID}})$$

  - + the digest is difficult to invert ⇒ the password can not be deduced;
  - − unprotected digests are subject to dictionary attacks;

- hashed with salt: unpredictable digests are stored (f = hash function h):

$$f(S_{\text{UID}}) = h(P_{\text{UID}} || \text{salt})$$

  - + dictionary attacks, even with rainbow tables, are made practically impossible.

4.2.1 Dictionary attack

> **dictionary attack** an attack that uses a brute-force technique of successively trying all the words in some large, exhaustive list

A **dictionary attack** is a kind of brute-force attack with the purpose of discovering a password by knowing its hash (= digest): a list, called **dictionary**, containing only the most probable passwords, that is the ones which humans choose more frequently because easy to remember.

Hypothesis The attacker has access to information in the server:

- he has the server database containing the hashes corresponding to users' passwords;
- he knows the hash algorithm used to compute hashes.

Pre-computation For every word p included inside a dictionary, the attacker:

```
for (each p in Dictionary)
   store(DB, p, h(p))
```

1. computes the hash h of the word;
2. stores the $\{p, h\}$ pair into a database:

Attack For every hash HP in the server database, the attacker performs an exhaustive lookup in the pre-computed database:

```
for (each HP in ServerDB)
   for (each (p, h) in DB) // lookup in the database
      if (h == HP) // password found
         write("password = " + p)
         go to next HP
```

Cases of study

- MySQL: starting from version 4.1, MySQL uses a **double hash** with SHA1 (without salt) ⇒ the double hash is not enough to prevent dictionary attacks: the pre-computed database can just be created by applying the hash function twice;
- attack to LinkedIn (June 2012): since cybercriminals had no ready dictionaries available, they used **crowdsourcing**: they published the list of all hashes to a website, and asked site users to contribute to password cracking by running on their own PCs a program attached to the list.

4.2.2 Rainbow table

Rainbow tables are special pre-computed databases which allow to achieve a tradeoff between:

- memory: just a subset of hashes is saved into the pre-computed database;
- computational time: when the chain is found, the chain needs to be retraced in order to find the password.

Hypotheses

- the attacker has access to information in the server (like in the dictionary attack);
- for the sake of simplicity, all passwords are made up of 5 digits $\Rightarrow 10^5$ possible passwords exist;
- the attacker chooses to build rainbow tables made up of 100 rows $\Rightarrow$ each row represents a chain of 1000 passwords;
- the attacker chooses a reduction function r which, given a hash h, returns any password p (which not necessarily is matching hash h).

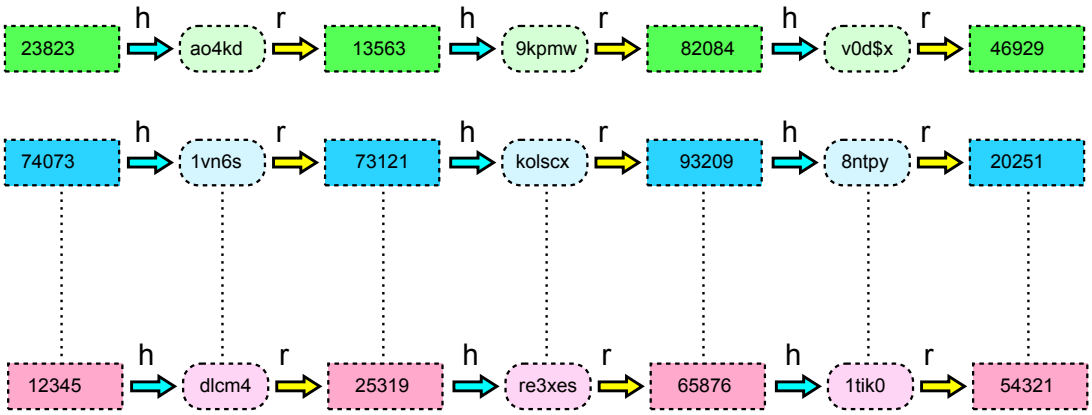


Figure 4.2: Example of rainbow table.[2]

Pre-computation The attacker:

```
for (100 distinct P)
    p = P
    for (i = 0 : 1000)
        k = h(p)
        p = r(k)
    store(RT, P, p)
```

1. he takes any 100 distinct passwords P among the 10^5 possible ones;
2. starting from each of these passwords, he builds a chain which crosses 1000 passwords by alternating hash h and reduction r functions;
3. he stores last password p (tail) achieved at the end of each chain, together with starting password P (head), into the rainbow table, which at the end will contain 10^2 $\{P, p\}$ pairs.

[2]This picture is derived from an image on Wikimedia Commons (Rainbow table1.svg), made by user Dake, and is licensed under the Creative Commons Attribution-Share Alike 4.0 International license.
[3]This picture is derived from an image on Wikimedia Commons (Rainbow table2.svg), made by user Dake, and is licensed under the Creative Commons Attribution-Share Alike 4.0 International license.

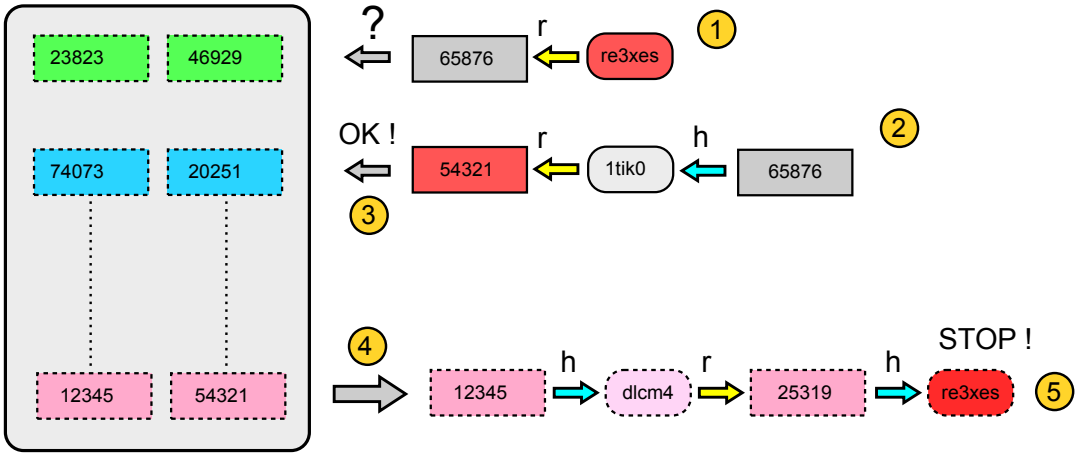


Figure 4.3: Example of dictionary attack with rainbow table (hash = `re3xes`, password = `25319`).[3]

Attack For every hash HP in the server database, the attacker:

```
for (each HP in ServerDB)
   for (i = 0 : 1000)
      h = (i == 0) ? HP : h(ph)
      ph = r(h)
      for (each (P, p) in RT) // lookup in the rainbow table
         if (p == ph) // chain found
            for (i = 0 : 1000) // lookup in the chain
               Ph = (i == 0) ? P : r(h)
               h = h(Ph);
               if (h == HP) // password found
                  write("password = " + Ph)
                  go to next HP
```

1. starting from hash HP, he builds a chain which crosses at most 1000 passwords by alternating reduction r and hash h functions;
2. he compares each intermediate password ph with tails p of all the chains in the rainbow table;
3. once the chain is found, he retraces the chain from head P to tail p analogously to pre-computation;
4. he compares each intermediate hash h with hash HP;
5. once hash HP is found in the chain, last password Ph on which the hash function has been applied is the user's password.

Improvements

- in order to avoid that the same password appears in multiple chains, different reduction functions $r_0() \dots r_n()$ can be used;
- pre-computed rainbow tables for various hash functions and password sets.

4.2.3 Password choice

Users tend to choose passwords which can be easily memorized, and hence simple to guess. Passwords should not be used at all, but using at least one password is unavoidable, unless biometric systems are used.

How to choose a strong password?

- it should use different kinds of characters (uppercase and lowercase alphabetic + digits + special characters);
- it should be as longer as possible (at least 8 characters);
- it should not be found in a dictionary;
- it should be periodically changed.

> **ransomware** a type of malware which restricts access to the computer system that it infects, and demands a ransom paid to the creator(s) of the malware in order for the restriction to be removed

Case of study attack to iCloud (May 2014): some cybercriminals violated some iCloud accounts protected by weak passwords, then they used the 'Find My Device' service to remotely put users' Apple devices into 'Lost Mode', asking for the payment of a ransom in order to unlock them.

4.2.4 Salt

> **salt** a data value used to vary the results of a computation in a security mechanism, so that an exposed computational result from one instance of applying the mechanism cannot be reused by an attacker in another instance

The **salt** is a number used to vary the password before computing the hash.

Salted hashing

1. a number:
   - random: it varies the password in an unpredictable way (better if with little used or control characters);
   - long: it increases the dictionary complexity by compensating the usage of short passwords;
   - nonce: it is different for each UID $\Rightarrow$ if two users are using the same password, their hashes will be different;

   is chosen as the salt;
2. hash HP is computed on the password concatenated to the salt:
   $$\text{HP} = \text{hash}\ (\ \text{password}\ ||\ \text{salt}\ )$$
3. the salt is stored in clear on the server too, in order to be able to later verify the password:
   $$\{\ \text{UID},\ \text{HP}_{\text{UID}},\ \text{salt}_{\text{UID}}\ \}$$

Advantages

- if two users are using the same password, their hashes will be different $\Rightarrow$ one can not discover to have the same password as the other one;
- dictionary attacks, even with rainbow tables, are made practically impossible:
  - a pre-computed dictionary for each possible salt would be needed;
  - if the user periodically changes his password (and hence the salt), the attacker will not have the time to create the pre-computed database.

KDF

A cryptographic key should be altogether random, that is each bit must have 50% probability to be 0 or 1, to be unpredictable and hard to guess. However human beings are not in general very good in thinking up random data.

Hash algorithms, besides guaranteeing data integrity, can be used also to generate random cryptographic keys starting from a secret password ⇒ the user does not need to deal with a sequence of bits without any meaning, while a password, even little random, can be easily memorized and typed by the keyboard.

A **Key Derivation Function** (KDF) is able to automatically derive one or more secret keys from a segret password by using a hash function:

$$K = \text{KDF}\,(P, S, I)$$

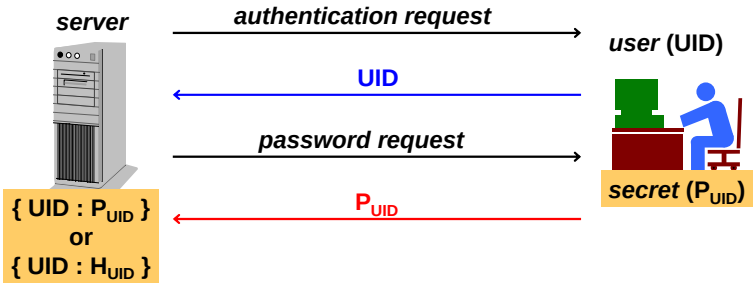
where:

- K is the set of the generated cryptographic keys;
- P is the password/passphrase provided as the input by the user;
- S is the salt, distinct based on the user;
- I is the number of iterations of the base function, in order to slow down the computation and make a possible attack more complicated.

The most used KDFs are based on cryptographic hash functions:

- PBKDF2: it uses SHA-1 ($|S| \geq 64$ bit, $I \geq 1000$);
- HKDF: it is based on HMAC.

4.3 Authentication based on reusable passwords



What the user knows is a **reusable password**: the user can re-use it as many times as he wants.

Authentication procedure

1. identification: the client sends its own UID;
2. proof request: the server asks for a password;
3. proof: the client transmits the password in clear (F = identity function I):

$$F\,(S_{\text{UID}}) = P_{\text{UID}}$$

4. check: the server:
   (a) computes the salted hash on the received proof;
   (b) compares the computed hash with the stored digest:

$$\text{hash}\,(\text{proof}\,||\,\text{salt}_{\text{UID}}) = \text{HP}_{\text{UID}}?$$

Disadvantages

- client side: the responsibility of preserving the password is left to the user:
  - the user could write the password in an unsafe place (e.g. post-it note);
  - users tend to choose passwords which can be easily memorized, and hence simple to guess;
- network side:
  - sniffing attacks: the user's secret is transmitted in clear;
  - replay attacks: the password is valid for multiple logins;
- server side: the salt should be used to protect the passwords stored on the server ⇒ the verification requires a digest computation for every proof.

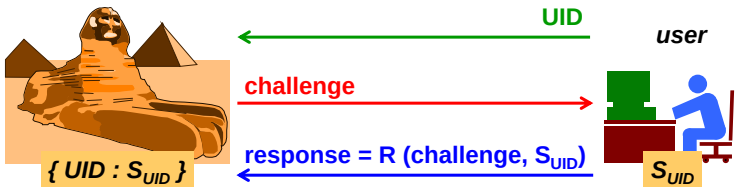
4.3.1 Challenge-response systems

> **challenge-response** an authentication process that verifies an identity by requiring correct authentication information to be provided in response to a challenge [...]

In order to avoid that the proof is intercepted during its transmission, **challenge-response systems** need to be used:

- symmetric (sect. 4.4): the user solves the challenge by computing its keyed-digest by using the secret shared with the server;
- asymmetric (sect. 4.5): the user solves the challenge by decrypting it by using his own private key.

4.4 Symmetric challenge-response systems



What the user knows is a shared secret: the user proves to know the secret without transmitting it, but using it to perform a computation.

Authentication procedure

1. identification: the client sends its own UID;
2. proof request: the server:
   (a) generates a nonce and random number;
   (b) sends the generated number as a **challenge**;
3. proof: the client:
   (a) computes the keyed-digest on the received challenge, by using the shared secret as the key (F = hash function R):

$$F(S_{\text{UID}}) = R(\text{challenge}, S_{\text{UID}})$$

(b) transmits the computed keyed-digest as the solution to the challenge;

4. check: the server:
   (a) performs the same solution computation by using the generated challenge and the shared secret;
   (b) compares the received proof with the computed solution:

$$\text{proof} = R\,(\text{challenge}, S_{\text{UID}})?$$

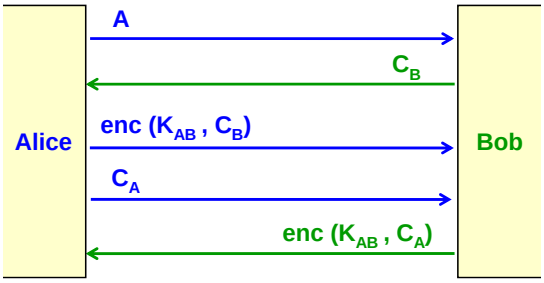
Advantages

- no sniffing attacks: the user's secret can not be deduced;
- no replay attacks: the challenge includes a nonce number.

Disadvantages

- client side: the user should have a hash function available to perform the solution computation ⇒ if he is using a public terminal and has not a hash function, he will not be able to reply to the challenge;
- server side: the server can not store password hashes, but needs to know in clear the secret in order to be able to perform the same solution computation.

4.4.1 Mutual authentication with symmetric challenge-response protocols



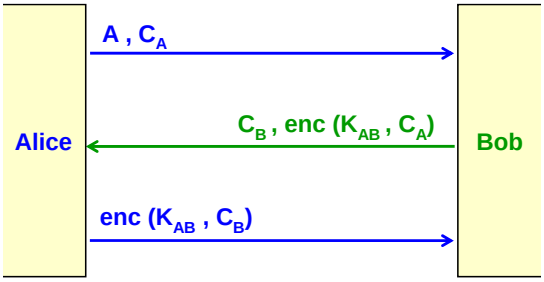
Symmetric challenge-response systems lend themselves well to do **mutual authentication** between two peers, where the server (Bob) is authenticated too:

1. Alice starts the exchange by sending her own UID;
2. Bob sends to Alice a challenge C_B;
3. Alice sends the solution to challenge C_B computed by using common key K_{AB};
4. Alice sends to Bob a challenge C_A;
5. Bob sends the solution to challenge C_A computed by using common key K_{AB}.

Tweak

Performance (but not security) can be improved by decreasing the number of exchanged messages:

1. Alice starts the exchange by sending her own UID, together with her challenge C_A;
2. Bob sends the solution to challenge C_A, together with his challenge C_B;
3. Alice sends the solution to challenge C_B.

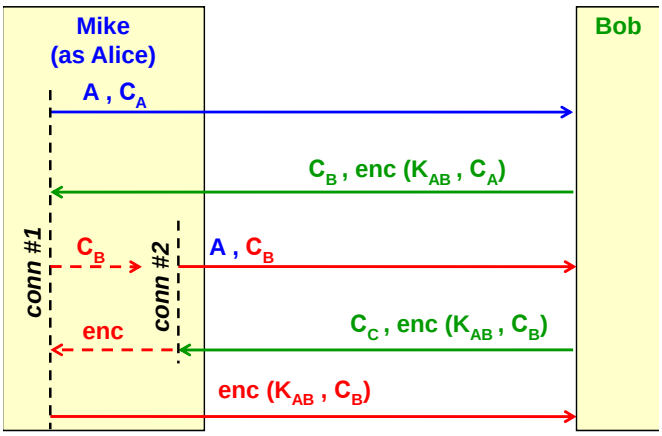


4.4.2 Possible implementation mistakes

Symmetric challenge-response protocols are easy to implement, but they may be subject to attacks if implemented in a wrong way:

- **invertible hash function**: an attacker could deduce the secret by intercepting the challenge and the solution:
  - solution: function R used to compute the solution should be a cryptographic hash function;
- **repeated challenge**: an attacker could repeat the response (replay attack):
  - solution: the challenge should be a nonce, so that solutions will be different on every user authentication;
- **predictable challenge** (e.g. based on date and time): an attacker could learn the solution of the next challenge by making the client authenticate to a fake server:
  - solution: the challenge should be random;
- **double authentication**: an attacker could learn the solution of the current challenge by making the server authenticate twice:
  - solution 1: the server should allow one connection at a time to be opened;
  - solution 2: the common key can be replaced by two distinct, one-way keys.

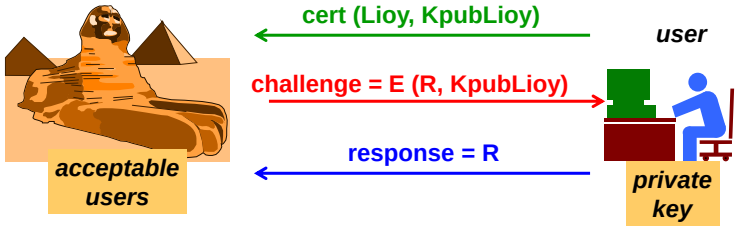
Attack with double authentication



Mike wants to authenticate himself to Bob by pretending to be Alice, without knowing the shared secret (that is common key K_{AB}):

1. Mike starts the exchange, by sending Alice's UID together with a challenge S_A;
2. Bob sends the solution to challenge S_A, together with a challenge S_B;
3. Mike starts a second exchange, by sending Alice's UID together with challenge S_B of which he does not know the solution;
4. Bob sends the solution to challenge S_B (together with a challenge S_C);
5. Mike sends the just received solution to challenge S_B of the first exchange, computed not by him but by Bob.

4.5 Asymmetric challenge-response systems



What the user knows is his own private key: the user proves to know the key without transmitting it, but using it to perform a computation. The private key is not shared: the server can perform the check by using the user's public key.

Authentication procedure

1. identification: the client sends its own public-key certificate;
2. proof request: the server:
   (a) generates a nonce number R;
   (b) encrypts generated number R by using the public key taken from the certificate;
   (c) sends the encrypted number as a challenge;
3. proof: the client:
   (a) decrypts the received number by using its own private key;
   (b) transmits the number in clear as the solution to the challenge;
4. check: the server compares the received proof with generate number R:

$$\text{proof} = R?$$

Advantages

- no sniffing attacks: the user's private key can not be deduced;
- no replay attacks: the challenge includes a nonce number;
- no confidentiality: no confidential information is stored on the server.

Disadvantage computational complexity on client side

4.5.1 Possible implementation mistakes

- **unprotected list of authorized users**: an attacker could add his name to it, so as to access by using his own key:
  - solution: the integrity and the authenticity of the list of authorized users stored on the server should be ensured;
- **fake certificate**: an attacker could create a fake certificate binding his own public key with the identity of one of the authorized users, so as to access by using his own key:
  - solution: all the certificates of the CAs along the certification path from the issuer CA up to a trusted root CA should be got, in order to check the validity of the digital signature of the received public key certificate;
- **broken name constraint**: an attacker could convince or manipulate a trusted CA to issue certificates outside its own domain (e.g. the CA of the Politecnico di Milano is certifying a user of the Politecnico di Torino):
  - solution: the certificate of the issuer CA should be got, in order to check that the certified user's name is included within the domain of trustworthy names ('name constraints' extension in the CA certificate[4]);
- **unaware signing** (with RSA keys): if an attacker sends as a challenge the cleartext digest of a document, the client will return the digest 'decrypted' by its private key which is equivalent to the digest signed by its private key ⇒ the client has unwillingly signed the document.[5]
  - solution: two different keys for signing and encryption should be used ('key usage' extension in the user's certificate[6]).

4.6 Authentication based on one-time passwords

> **one-time password** a simple authentication technique in which each password is used only once as authentication information that verifies an identity [...]

What the user knows is a **one-time password** (OTP), that is non-reusable: it is valid just for a single login.

OTP systems can be:

- asynchronous (sect. 4.6.1): the password does not depend on time (e.g. S/KEY: sect. 4.7);
- synchronous (sect. 4.6.2): the password depends on time (e.g. SecurID: sect. 4.8).

Advantages

- no sniffing attacks: the user's secret is not transmitted;
- no replay attacks: the password is valid just for a single login.

[4] Please see section 3.2.2.

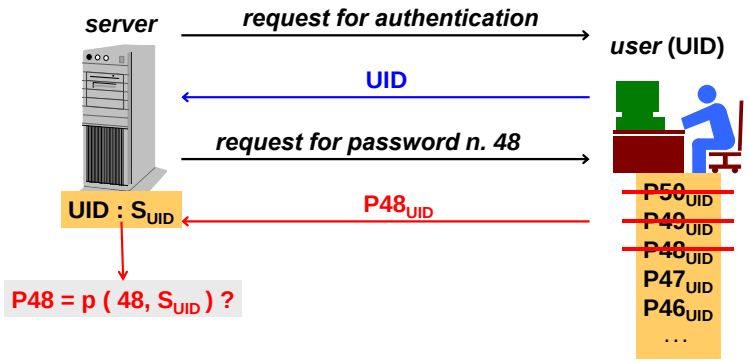
[5] The digital signature would theoretically require the decryption of the digest, but with RSA keys the encryption operation by using the private key is identical to the decryption operation by using the same private key, because both operations consist in exponentiation by the same private exponent (an analogous consideration is valid for the public key).

[6] Please see section 3.2.2.

Disadvantages

- manual use: passwords usually need to be entered by a human;
- uncomfortable use: entering a different password on every login is uncomfortable, especially if logins are periodic (e.g. e-mail);
- security: passwords generated by a machine are pseudo-random;
- provisioning: the user can not remember all possible passwords.

4.6.1 Asynchronous OTP systems



The password is taken by a list of one-time passwords first provided to the client by the server, which generates it starting from a secret not shared with the user.

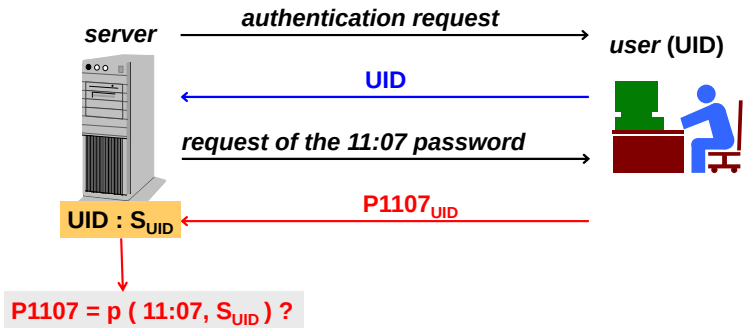
Authentication procedure

1. identification: the client sends its own UID;
2. proof request: the server sends a nonce number n;
3. proof:
   (a) the user searches the list for the password at the n-th position and enters it;
   (b) the client transmits the password in clear;
4. check: the server:
   (a) computes the n-th password starting from user's secret S_{UID};
   (b) compares the received proof with the computed password:

$$\text{proof} = p\,(n, S_{\text{UID}})?$$

4.6.2 Synchronous OTP systems

The password is computed on the fly by the server and the client starting from the current date and time, together with the user's shared secret.



Authentication procedure

1. identification: the client sends its own UID;
2. proof request: the server asks for a password;
3. prova: the client:
   (a) computes the password by generation function p starting from user's secret S_{UID} and the current date and time (e.g. through a token);
   (b) transmits the password in clear;
4. check: the server:
   (a) computes the password by generation function p starting from user's secret S_{UID} and the current date and time;
   (b) compares the received proof with the computed password:

$$\text{proof} = p\,(\text{date and time}, S_{\text{UID}})?$$

Disadvantages

- cost: due to hardware authenticators;
- synchronization: since passwords are generated based on the date and time, time synchronization between client and server is important;
- DoS attacks: an attacker could make the server block the legitimate user's account by performing a certain number of consecutive failed login attempts:
  - palliative: increasing delays between login attempts could be introduced to make the attacker 'tired';
- attacks with social-engineering techniques: an attacker could call, pretending to be the legitimate user, and declare the token as being lost and ask for a new one to be initialized.

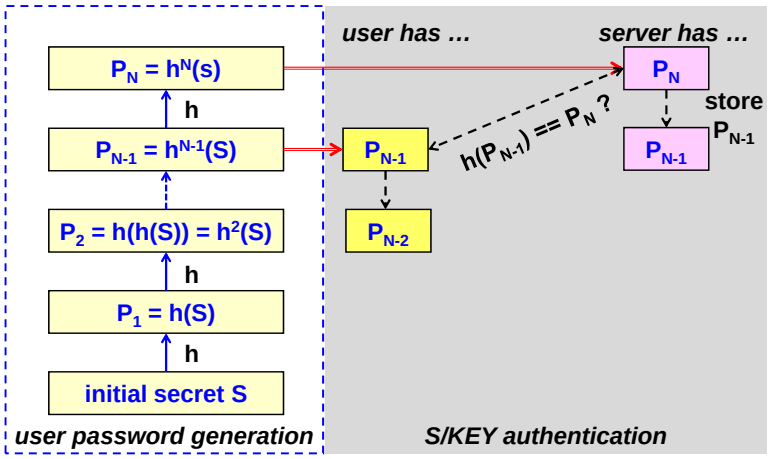
4.6.3 Provisioning ways

How to provide the user with the list of one-time passwords?

- for insecure clients (e.g. Internet café) or clients without autonomous computational capability (e.g. terminal):
  - sheet of paper on which the list of passwords pre-computed by the server;
  - hardware authenticator (sect. 4.8.2): it is able to replicate the same computation performed by the server;

- for secure clients with autonomous computational capability (e.g. user's personal device):
  - software authenticator (sect. 4.8.2): an ad-hoc application, installed on the user's client, able to replicate the same computation performed by the server.
    The application could be possibly integrated into the communication software (e.g. Telnet, FTP) or hardware (e.g. router) for an automatic authentication without the user's manual intervention.

4.7 S/KEY system



The **S/KEY** system was the first OTP system, invented in Bell Labs. Its release into the public domain (RFC) allowed the birth of several variants, including some commercial implementations with authenticators.

Authentication procedure

1. the S/KEY system on the client asks the user to enter a secret, at least 8-character-long passphrase PP, to simplify the password generation;
2. the S/KEY system concatenates the passphrase with a seed sent in clear by the server, making secret $S \Rightarrow$ the same passphrase can be used for different servers by changing the seed;
3. the S/KEY system computes N passwords by repeatedly applying a hash function h to secret S, and stores them:

$$P_1 = h(S)$$
$$P_2 = h(P_1) = h(h(S))$$
$$\dots$$

   Originally the MD4 algorithm was used (the choice of a stronger algorithm is possible): each password is 64 bits long and is extracted from a 128-bit-long hash;
4. the S/KEY system initializes the authentication server with last password P_N: this password will never be used directly for authentication, but just indirectly to check other passwords;
5. the server, on the first login, prompts for password P_{N-1};

6. the server checks whether password P_{N-1} is right by using password P_N:
$$h(P_{N-1}) = P_N?$$
7. if password P_{N-1} is right, the server stores it in order to be able to check password P_{N-2} on the second login.

Advantages

- no sniffing attacks: hash functions are not invertible ⇒ the attacker can not restore the chain, unless he learns the user's secret;
- no replay attacks: passwords are one-time;
- no secrets on the server: the server does not need to know the user's secret: only the client knows all passwords ⇒ storing confidential information on the server is not a problem;
- ease of use: each password can be easily entered by the user as a sequence of 6 English words, converted into bits thanks to a dictionary of 2048 words.

4.8 RSA SecurID

token a data object or a physical device used to verify an identity in an authentication process

The **SecurID** system was invented and patented by RSA.

What the user owns is an authenticator, called **token**:

- it stores the user's shared secret safely;
- it is able to generate one-time passwords depending on time (synchronous OTP mechanism):
$$P_{\text{UID}}(t) = h(S_{\text{UID}}, t)$$

The system is based on a proprietary, secret hash algorithm ⇒ this is an example of security through obscurity: it can not be certain that it is actually strong against possible attacks.

4.8.1 Authentication protocol

SecurID is a two-factor authentication:

- **token-code**: a random, non-reusable access code, made up of 8 digits, computed every 60 seconds by the token as a function of the user's secret (seed) and the current date and time;
- **PIN** associated to the token: if somebody steals the token, he can not authenticate himself on behalf of the legitimate owner.

Once the user's identity has been checked based on username (UID) and PIN, the server computes token-code TC_0 based on the current minute and compares it with the received token-code. However client and server could not be synchronized ⇒ if the first check fails, the server will compute token-codes related to previous (TC_{-1}) and next (TC_{+1}) minutes, too. After a certain number (default: 10) of consecutive failed authentication attempts, the user's account will be blocked ⇒ this makes DoS attacks possible.

As a further security measure, a **duress code** is associated to every authenticator: if a bad guy is forcing the user to authenticate under threat, he can type the duress code as the PIN:

- the authentication will appear to be successful;
- the performed operations will be not actually applied;
- an alert is launched to the administrator.

Each single token can have three different login keys to be used with three different servers.

4.8.2 Authenticators

Hardware authenticators

Hardware authenticators are used for:

- insecure clients (e.g. Internet café);
- clients without autonomous computational capability (e.g. terminal).

Hardware authenticators differ into four categories:

- **SecurID Card**: classic card which simply shows the token-code corresponding to the current date and time, but the PIN needs to be entered on a computer keyboard (dangerous on insecure clients);
- **SecurID PinPad**: it has an integrated keypad by which the user can enter the PIN; if the PIN is right the device will generate a token-code*, that is a token-code version which already embeds the PIN $\Rightarrow$ only the username and the token-code* are transmitted over the network;
- **SecurID Key Fob**: device in key-fob format, so the user will always bring it with himself;
- **SecurID Key Fob and smart-card**: it embeds all the features of previous devices.

Disadvantage Since the internal quartz clock is subject over time to an intrinsic shift (at most 15 seconds per year), a hardware authenticator has a maximum validity equal to 4 years and should be replaced.

Software authenticators

Software authenticators are installed on secure clients with autonomous computational capability (e.g. user's personal device), and implement in software the same computation algorithm of hardware algorithms with the purpose to reduce costs.

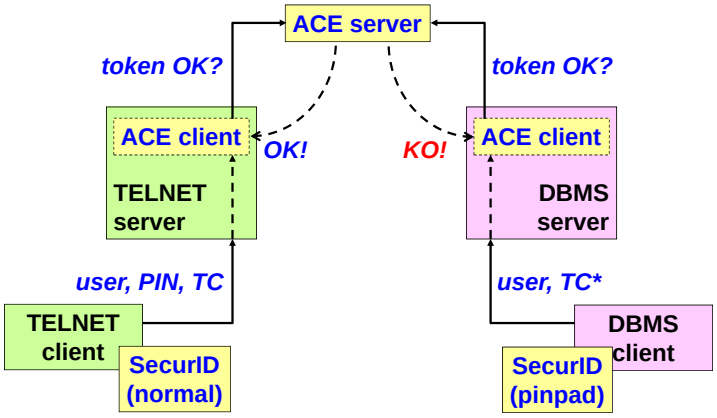
Disadvantage Clock synchronization becomes very significant: the date and time is no longer given by a quartz clock but is provided by the operating system on the user's device:

- computer: the date and time can be synchronized via the Internet through Network Time Protocol (NTP);
- mobile phone: it can take the date and time straight from the cellular network.

4.8.3 Authentication architecture

Nowadays a company has no longer a single server which the user directly interacts with through the authentication protocol, but multiple servers which should share the same way of authentication: the whole authentication management is centralized on a single authentication server, which all application servers contact in order to check the user's identity:

- application client: it directly interacts with an application server through an application protocol (e.g. Telnet, SQL), and can authenticate by means of a variety of SecurID authenticators (e.g. Card, PinPad);
- application server: a software, called **client Access Control Engine** (ACE), is installed which contacts, through an encrypted channel, the ACE server whenever the user's identity needs to be checked;
- authentication server: a software, called **server ACE**, is installed which is able to interpret the authentication information by storing/computing user's lists, token-codes, secrets and PINs.

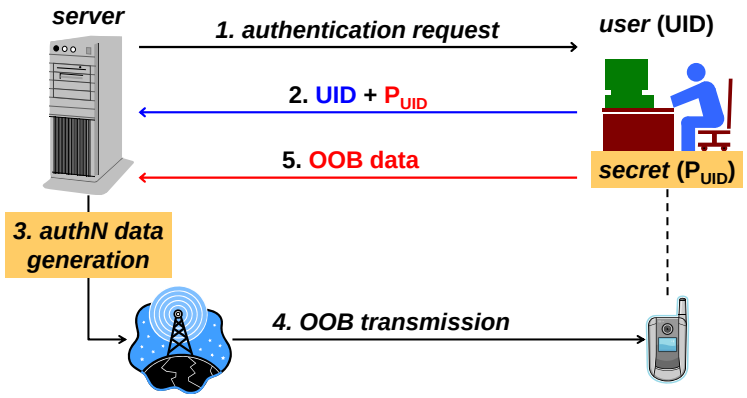


Advantages

- the user does not have to remember different passwords to access different services;
- the ACE server also includes a SQL interface to access a database storing user's data, useful for an old organization with a pre-existing database.

Disadvantage All application servers have to trust the authentication server.

4.9 Out-of-band authentication



Out-of-band (OOB) authentication is based on both username and password, plus some additional data sent over another communication channel (e.g. via SMS) thus providing an additional level of security.

4.10 Biometric systems

> **biometric authentication** a method of generating authentication information for a person by digitizing measurements of a physical or behavioral characteristic [...]

How can we be sure of interacting with a human and not a computer program (e.g. sending a password stored in a file)?

- **CAPTCHA techniques** (acronym of 'Completely Automated Public Turing test to tell Computers and Humans Apart'): the user is asked to read something only a human is able to correctly interpret (e.g. images of distorted characters, disturbed sounds);

- **biometric techniques**: they measure a biologic characteristic of the user's (e.g. fingerprints).

Biometric systems rely on the fact that each individual has some biologic unique characteristics:

- fingerprint;
- voice;
- retinal scan;
- iris scan;
- hands' blood vein pattern.

Biometric systems are good for a local use, not on the Internet: if somebody steals biometric data, the victim will not be able to change his biologic characteristic (unlike a password-based authentication system).

Issues

- psychological acceptance: users are afraid of being filed or damaged by intrusive technologies;
- accuracy: biometric systems have a certain unavoidable margin of error;
- interoperability: each device offers its own interfaces to application developers ⇒ API/SPI standards are needed.

4.10.1 FAR and FRR

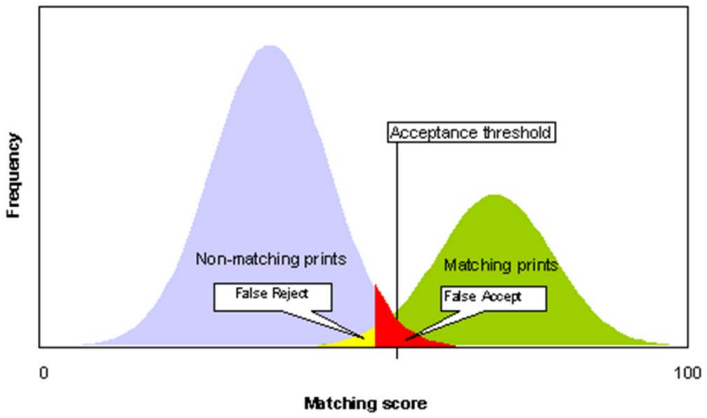


Figure 4.4: The area where Gaussian curves overlap is the uncertainty area.

An individual's physical characteristics are variables:

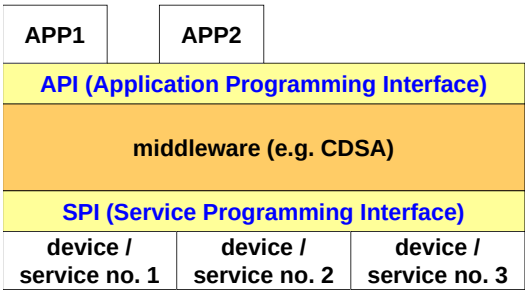
- a finger wound may affect the fingerprint;
- excitement may make voice trembling;
- alcohol and drug may alter the retinal blood pattern.

The quality of a biometric system is measured based on two statistical parameters, depending on the device cost:

- **False Acceptance Rate** (FAR): it measures how many times authentication is successful when the user is no the registered one (false positive);
- **False Rejection Rate** (FRR): it measures how many times authentication fails when the user is the registered one (false negative).

Positioning the **acceptance threshold** depends on the context: in military applications, the acceptance threshold has to be positioned so as to always reject when being in the uncertainty area, even causing a lot of false negatives.

4.10.2 CDSA



A standard, unified API/SPI, based on the CDSA system, is being developed:

- Application Programming Interface (API): it offers to application developers a common interface to communicate with biometric devices even by different vendors;
- middleware (e.g. CDSA): it automatically maps API calls to SPI calls;
- Service Programming Interface (SPI): it offers to biometric system vendors a common interface to communicate with applications even by different developers.

4.11 Kerberos

Kerberos is an authentication system developed at MIT as a part of the Athena project.
Every user is associated with:

- **credential**: it uniquely identifies the user and is in the form:
  user.instance@realm
  - **user**: it is the username (UID);
  - **instance**: it is the user's role, that is the privileges he has (e.g. simple user, system administrator);
  - **realm**: it is the Kerberos domain, that is the set of systems which use Kerberos as the authentication system.
- **password**: it is the shared secret between the user and the authentication server (a user can use the same password for multiple instances).

4.11.1 Data formats

The **ticket** is a data structure, generated by a trusted third party (TTP) and encrypted by key K_S of the server it is meant for, with which the client authenticates itself to the server the ticket is meant for:

- `server-id`: the ID of the server the ticket is meant for;

(a) ticket *(b) authenticator*

Figure 4.5: Data formats in Kerberos version 4.

- `client-id`: the user's credential;
- `client-address`: the IP address of the client;
- `timestamp`: the date and time where the ticket was generated;
- `life` (max 21 hours): the ticket lifetime, variable depending on the task it has been released for, after which the ticket becomes no longer valid;
- $K_{S,C}$: the symmetric **session key** by which the authenticator along with the ticket has been encrypted; it is not negotiated between the client and the server, but is generated on the fly by the TTP.

The ticket represents, for the server it is meant for, the client authentication proof:

- the TTP which generated it guarantees that the client has proved its identity;
- the ticket is provided by the TTP to the client encrypted by a key which only the legitimate client knows;
- the ticket is encrypted by the key of the server it is meant for $\Rightarrow$ it can not be changed by the client (it is an **opaque structure**, a sort of binary blob).

The **authenticator** is a data structure, generated by the client and encrypted by session key $K_{S,C}$, which the client sends along with the ticket:

- `client-id`: the user's credential;
- `client-address`: the IP address of the client;
- `timestamp`: the date and time where the authenticator was generated.

Security measures

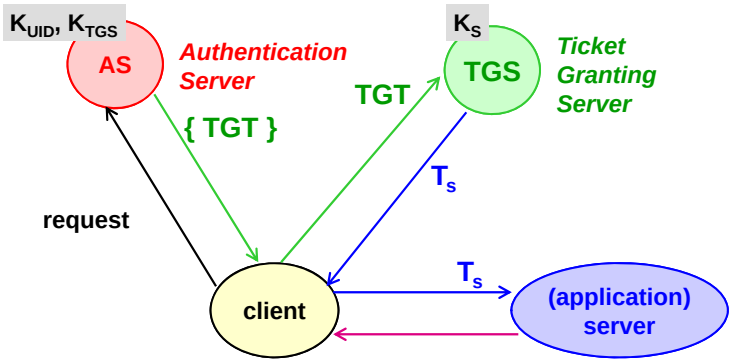
- against sniffing attacks: all communications are encrypted by symmetric keys never transmitted in clear:
  - the user's password and server keys are never transmitted, but they are just used locally as symmetric cryptographic keys;
  - the session key is transmitted by the TTP to the client in an encrypted form;
  - the session key is transmitted by the client to the server in an encrypted form, included inside the ticket;
- against replay attacks: the authenticator has the purpose of protecting against replay attacks:
  - the ticket and the authenticator are bound to the client through its IP address;

- the ticket and the authenticator are bound together through the session key;
- the authenticator has a very short lifetime: the timestamp included in the authenticator can not be too 'old', and can not be changed because it is encrypted;
- the authenticator can be used only once: it is encrypted by a session key which can be used only once.

Changes in version 5

- cryptographic algorithm: the choice of the symmetric cryptographic algorithm is no longer constrained to DES only;
- ticket lifetime: tickets can have unlimited lifetime, because the dates and times of validity start and end are explicitly written (although it is recommended to always limit ticket lifetimes);
- inter-realm authentication: a ticket issued inside a realm becomes valid even in another realm;
- forwardable tickets: also another client with another IP address can use the ticket;
- extended tickets: additional information depending on the service provider can be inserted into the ticket.

4.11.2 Authentication procedure



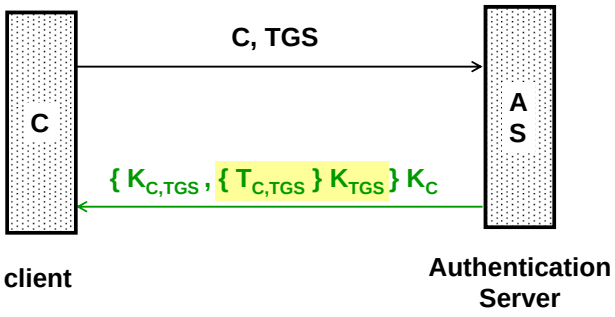
Kerberos distinguishes three entities from the logical point of view:

- client: on every login the user locally types his password K_{UID};
- **Authentication Server** (AS): it is in charge of:
  - storing a copy of all user's passwords K_{UID} and TGS key K_{TGS} in a secure way;
  - checking the client credential;
  - generating ticket TGT meant for the TGS;
- **Ticket Granting Server** (TGS): it is in charge of:
  - storing its key K_{TGS} and a copy of application server passwords K_S in a secure way;
  - checking the TGT ticket provided by the client as an authentication proof;
  - generating ticket T_S meant for the application server;
- application server: it is in charge of:
  - storing its key K_S in a secure way;

– checking ticket T_S provided by the client as an authentication proof;
– in turn authenticate itself to the client in case of mutual authentication;
– providing the application service (e.g. network printer, remote disk).

The AS and the TGS form the **trusted third party** (TTP).

TGT request

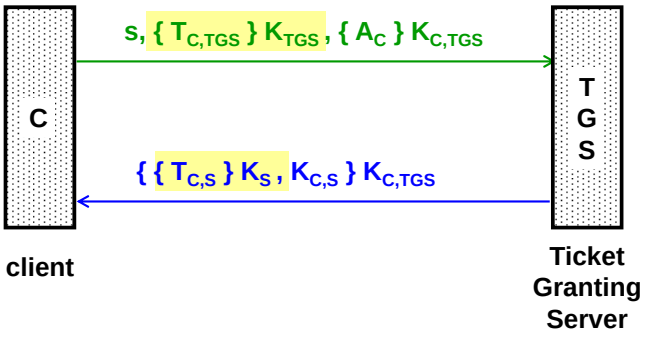


The client asks the AS for a ticket meant for the TGS:

1. the client sends to the AS in clear:
   - its credential C to identify itself to the AS;
   - the ID of the TGS it wants to contact;
2. the AS sends back to the client:
   - the ticket $T_{C,\text{TGS}}$:
     – it is encrypted by key K_{TGS} of the TGS the ticket is meant for: the ticket is opaque for the client;
     – it includes a copy of session key $K_{C,\text{TGS}}$ between the client and the TGS: the TGS should know the session key in order to be able to later decrypt the authenticator;
   - session key $K_{C,\text{TGS}}$ between the client and the TGS: the client should know the session key in order to be able to later encrypt the authenticator;

   both of them encrypted by client key K_C: only the legitimate client knows the key to decrypt them.

Ticket request

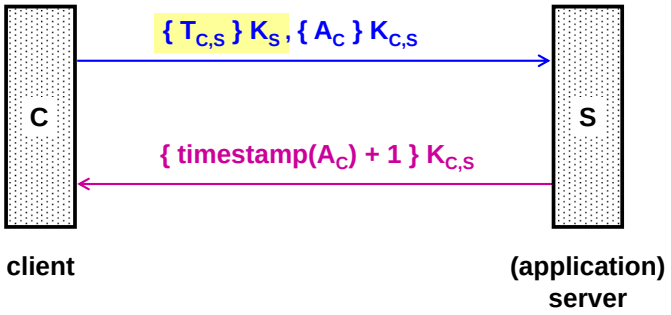


The client uses the ticket provided by the AS to ask the TGS for a ticket meant for the application server:

3. the client sends to the TGS:
   - the ID s, in clear, of the application server it wants to contact;
   - ticket $T_{C,\text{TGS}}$ it has received from the AS, encrypted by TGS key K_{TGS}: it represents for the TGS the proof that the client owns key K_C, by which it could decrypt ticket $T_{C,\text{TGS}}$ (client authentication);
   - authenticator A_C, encrypted by session key $K_{C,\text{TGS}}$ it has received from the AS: it will be able to be decrypted by the TGS by taking session key $K_{C,\text{TGS}}$ from ticket $T_{C,\text{TGS}}$;
4. the TGS sends back to the client:
   - ticket $T_{C,S}$:
     - it is encrypted by key K_S of the application server the ticket is meant for: the ticket is opaque for the client;
     - it includes a copy of session key $K_{C,S}$ between the client and the application server: the application server should know the session key in order to be able to later decrypt the authenticator;
   - session key $K_{C,S}$ between the client and the application server: the client should know the session key in order to be able to later encrypt the authenticator;

   both of them encrypted by session key $K_{C,TGS}$: only the client authenticated to the AS knows the key to decrypt them.

Ticket usage



The client uses the ticket provided by the TGS:

5. the client sends to the application server:
   - ticket $T_{C,S}$ it has received from the TGS, encrypted with application server key K_S;
   - authenticator A_C, encrypted by session key $K_{C,S}$ it has received from the TGS: it will be able to be decrypted by the application server by taking session key $K_{C,S}$ from ticket $T_{C,S}$;
6. the application server sends to the client the timestamp +1 of authenticator A_C, encrypted by session key $K_{C,S}$: it represents for the client the proof that the application server owns key K_S, by which it could decrypt the ticket from which it took the key by which to decrypt the authenticator (server authentication).

4.11.3 Advantages

- versatility: a lot of services can be 'kerberized', that is adapted to use the ticket mechanism: K-POP, K-NFS (for network disks), K-LPD (for network printers), K-telnet, K-ftp, K-dbms;
- single login for all kerberized services: one ticket can be used for multiple kerberized services;
- intermittent connections: the ticket keeps being valid even if the client disconnects from the network and connects again to it, avoiding continuous login requests;
- commercial support: Microsoft adopted Kerberos*, a proprietary implementation by Kerberos, starting from Windows 2000.

4.11.4 Issues

- storage of secrets on the server: the AS stores all user's passwords and the TGS key in clear, while the TGS stores application server keys in clear;
- IP spoofing attacks: the ticket and the authenticator are bound to the client through its IP address;
- DoS attacks: an attacker can overload a Kerberos server by performing a lot of ticket requests;
- clock synchronization: the authenticator can no longer be valid if the time spent in order to go from the client to the server is too long:
  - LAN: it is useful because in case of attack it is important to be able to trace back to what happened and correlate events for auditing purposes;
  - WAN: delays may be too high due to longer distances ⇒ Kryptonight is a Kerberos implementation developed by IBM which does no longer need timestamps;
- dial-up: if the user connects from remote (e.g. home), the password should be transmitted in clear across the network until his kerberized workstation ⇒ an encrypted channel or other authentication mechanisms are needed.

4.12 SSO

Single Sign-On (SSO) providing the user with a single 'credential' by which to authenticate himself for all operations on any system

Kerberos is an example of a more general concept, **Single Sign-On** (SSO):

- fake: single services require independent authentications with different passwords:
  - password wallet: it asks the user for a single global password to access the password database, and then it provides the right password to every service requiring authentication in a transparent way to the user;
- actual: all services actually share the same authentication system:
  - multi-application authentication technique (e.g. asymmetric challenge-response systems, Kerberos): all services are able to recognize the user's single credential (e.g. public key certificate, Kerberos ticket);
  - multi-domain SSO: a service accepts the credential (e.g. SAML token) from a service in another domain (e.g. Google).

4.13 Authentication interoperability

Proprietary implementations cause interoperability issues among authentication solutions from different vendors.

In recent years an initiative called **OATH** was born aimed at achieving interoperability for OTP, symmetric challenge-response and asymmetric challenge-response systems, by defining standards for the client-server protocol and the data format on the client. **Universal strong authentication** is achieved when all users can use devices from any manufacturer, satisfying OATH specifications, for any network service.

Some specifications have already been published in the RFC form:

- HMAC OTP (HOTP): one-time passwords based on HMAC digest;
- Time-based OTP (TOTP): one-time passwords based on time;
- OATH Challenge-Response Protocol (OCRA): challenge-response protocol;
- Portable Symmetric Key Container (PSKC): XML format for symmetric key transport;
- Dynamic Symmetric Key Provisioning Protocol (DSKPP): network protocol for symmetric key distribution from a key server.

Chapter 5

Security of IP networks

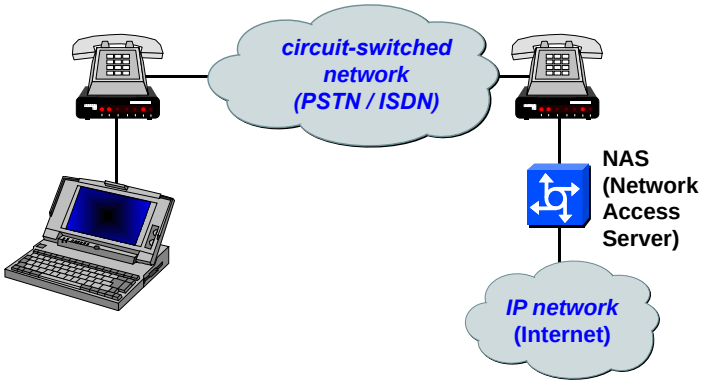
Security can be implemented at any layer in the OSI stack (except for the presentation layer, which just converts data): which is the optimal OSI layer at which to implement security?

In general an optimal layer can not be defined, but multiple solutions at various layers should be combined:

- higher layers: also more sophisticated attacks can be blocked:
  - \+ decisions are based on specific information (e.g. username, commands, data) ⇒ decisions are better;
  - − intruders can act undisturbed at underlying layers (e.g. DoS attacks);
- lower layers: it is better to just block the most basic attacks:
  - − data on which decisions are based are few (e.g. MAC address, IP address) ⇒ some good guys risk to be blocked and some bad guys risk to be allowed to pass;
  - \+ intruders can be 'expelled' more quickly.

5.1 Authentication for dial-up connections

5.1.1 Channel authentication



In the past internet connections were dial-up: the user connects through a modem, which directly communicated with the ISP modem across a point-to-point link over a data-link-layer (that is with no intermediate routers) circuit-switched network.

The user's modem and the ISP modem communicate by using **Point-to-Point Protocol** (PPP), which is able to encapsulate layer-3 (e.g. IP) packets and transport them over a point-to-point link, called **PPP channel**:

- physical (e.g. Public Switched Telephone Network [PSTN], Integrated Services Digital Network [ISDN]): there is a physical wire between the user's modem and the ISP modem through which PPP frames travel;
- virtual: there is a logical link (tunnel) between the user's modem and the ISP modem:
  - layer-2 (e.g. xDSL with PPPoE): PPP frames are encapsulated into Ethernet frames;
  - layer-3 (e.g. VPN with L2TP): PPP frames are encapsulated into UDP packets.

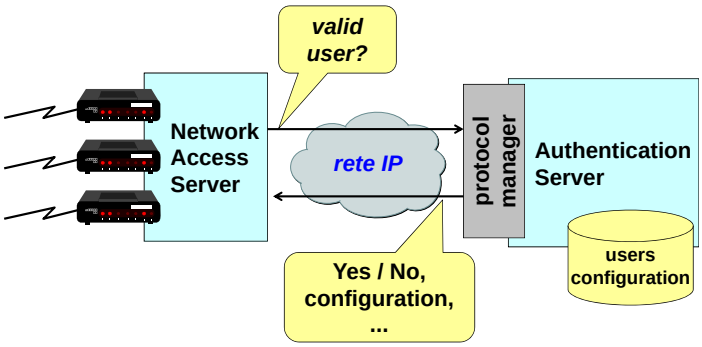
The PPP channel is enabled in three sequential steps:

1. connection (e.g. via Link Control Protocol [LCP]): the connection is opened and some parameters are negotiated;
2. authentication (e.g. via PAP, CHAP, EAP) (optional): the layer-2 PPP channel is authenticated, that is the user shows his identity;
3. encapsulation (e.g. via IP Control Protocol [IPCP]): PPP frames are encapsulated into layer-3 (e.g. IP) packets.

Three protocols mainly exist which allow to perform the authentication step of the PPP channel:

- PAP (Password Authentication Protocol): username and password are sent in clear;
- CHAP (Challenge Handshake Authentication Protocol): it uses a symmetric challenge-response authentication system, based on the user's password;
- EAP (sect. 5.2): the protocol is independent of the external authentication system (e.g. OTP, challenge-response, TLS).

5.1.2 Authentication check



The ISP **Network Access Server** (NAS) is the gateway toward the external network (e.g. Internet): its main task is to translate analog (telephone) signals into bits (IP packets).

In addition, the NAS should provide the three security features knows as 'the three As':

- **Authentication**: checking the identity claimed by the user based on his credentials (e.g. password, OTP);
- **Authorization**: checking which actions the user can perform and which services he can access;

accounting the process of tracking and/or analyzing user activities on a network by logging key data (e.g. amount of time in the network, services accessed, amount of data transferred)

- **Accounting**: keeping track of the user activity for various purposes (e.g. billing, auditing).

All information about the users allowed to the service could be directly inserted into the NAS, but if NASes are multiple and not close the user database should be concentrated to a single server, called **Authentication Server** (AS), to which all NASes are connected through a local IP network:

1. the NAS asks the AS to check whether the user is allowed to connect to the network;
2. the AS returns the check outcome, together if yes with the user's network configuration (e.g. proxy, DNS).

Each NAS communicates with the AS by using one of the following protocols:

- RADIUS (sect. 5.3): it is currently the most widespread protocol;
- DIAMETER (sect. 5.4): it is an evolution to RADIUS;
- TACACS+ (TACACS, XTACACS): originally it was technically better than RADIUS, but being a proprietary protocol by Cisco it is less widespread.

NASes can communicate with the AS by using different protocols one from each other ⇒ the **protocol manager** is in charge of supporting multiple authentication protocols at the same time.

5.2 EAP

Extensible Authentication Protocol (EAP) is a flexible data-link-layer authentication framework.

EAP pre-defines three kinds of authentication:

- MD5-challenge: challenge-response system similar to CHAP and based on MD5 keyed-digest;
- OTP;
- generic token card.

EAP is extensible: new kinds of authentication can be added:

- PPP EAP TLS authentication protocol: it adds TLS support;
- RADIUS support for EAP: it adds RADIUS support.

5.2.1 Encapsulation protocol

The authentication step takes place before the layer-3 link setup ⇒ EAP uses its own encapsulation protocol for authentication data:

- it is independent of the layer-2 protocol: it can authenticate not just PPP connections, but also connections such as 802.11 (wi-fi), 802.3 (Ethernet), 802.5 (token ring);
- it requires explicit ACKs/NACKs: every packet should be confirmed by an ACK, otherwise it is retransmitted up to 3/4 times;

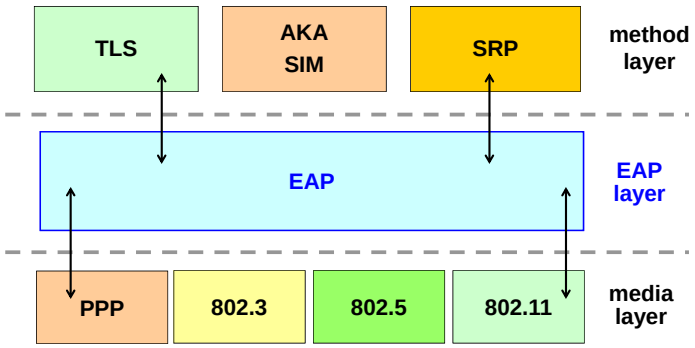
- it does not implement sliding-window mechanisms: it assumes that all packets always arrive in-sequence:
  - physical PPP channel: PPP guarantees that all packets always arrive in-sequence (single link);
  - virtual PPP channel: in UDP and IP packets may arrive out-of-sequence (multiple links);
- it does not support fragmentation: managing fragmentation if the payload is greater than the MTU is up to EAP methods.

5.2.2 Authentication methods

EAP assumes the physical link as insecure: no password is transmitted in clear, but the needed security services are provided by the chosen authentication methods:

- EAP-TLS: basic protocol for the Web;
- EAP-MD5: symmetric challenge-response system based on a MD5 keyed-digest for client (not mutual) authentication;
- EAP-TTLS: it establishes a secure TLS tunnel where to operate any EAP method (even username and password);
- EAP-SRP (Secure Remote Password);
- GSS_API: Kerberos system generalization;
- AKA-SIM (Authenticated Key Exchange): challenge-response system based on data taken from the SIM card in a mobile phone.

5.2.3 Architecture



From the architectural point of view three layers are distinguished:

- **media layer**: it is the underlying layer-2 protocol which transports frames;
- **EAP layer**: it is the EAP encapsulation protocol for the authentication data;
- **method layer**: it is the EAP authentication methods for the security service.

The EAP layer is a sort of middleware: any authentication method can be used with any layer-2 protocol.

5.3 RADIUS

The **Remote Authentication Dial-In User Service** (RADIUS) protocol is currently the de-facto standard for remote authentication.

Characteristics

- it implements a client-server scheme between NAS and AS;
- it allows user administration and accounting centralized in the AS;
- it allows network access through:
  - physical ports: analog, ISDN, IEEE 802;
  - virtual ports: tunnels, wireless accesses;
- it lies on the UDP protocol: it uses port 1812 for authentication and port 1813 for accounting (unofficial implementations based on ports 1645 and 1646 exist);
- it extends the UDP protocol: the packet is transmitted again when a timeout expires;
- it supports user authentication through PAP, CHAP and EAP protocols;
- attributes are in the **Type-Length-Value** (TLV) **format**: if a server does not know a type of attribute, it can 'skip' it thanks to the length field;
- the RADIUS server can act as a proxy toward other authentication servers: the access request by a user belonging to another domain is forwarded to the non-RADIUS authentication server for that domain, with its own database, delegating to it the authentication part:

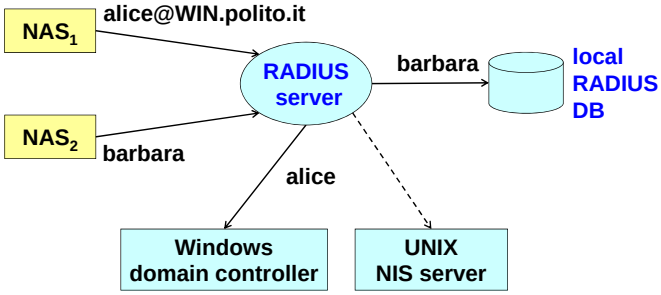
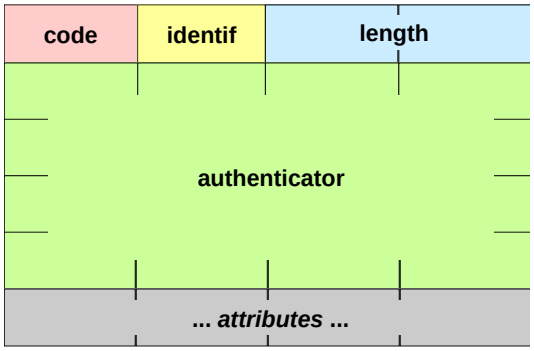


Figure 5.1: Alice's request access is forwarded to server 'Windows domain controller' belonging to domain `WIN.polito.it`.

5.3.1 Packet format

All RADIUS packets have the following format:

- Code field (1 byte): it identifies the type of packet:
  - 'Access-Request' value: it is the NAS request, and includes the access credentials (e.g. username and password);
  - 'Access-Accept' value: the AS tells that the access request has been accepted, and includes the network configuration parameters for the user's node;
  - 'Access-Reject' value: the AS tells that the access request has been rejected (e.g. due to wrong credentials);



  - 'Access-Challenge' value: the AS asks the user for additional information (e.g. PIN, token-code, secondary password);

- Identifier field (1 byte): it identifies the request;
- Length field (2 bytes): it specifies the length of the packet;
- Authenticator field (16 bytes): it contains the authenticator, and is computed in two different ways depending on the type of packet:
  - **Request Authenticator**: it is in NAS requests, and is made up of 16 random, nonce bytes generated by the NAS;
  - **Response Authenticator**: it is in AS responses, and is the result from a MD5 keyed-digest:

    md5(code || ID || length || Request Authenticator || attributes || key)

    * fields (code, ID, length, attributes): it provides integrity for the packet;
    * Request Authenticator: the response is tied to the request ⇒ it prevents replay attacks;
    * key: it is the secret shared with the NAS ⇒ it provides AS authentication against the NAS;

- **attributes**:
  - User-Name attribute (type 1): it contains a generic text, a Network Access Identifier (NAI) (= username[@realm]) or a distinguished name (DN);
  - User-Password attribute (type 2): it contains the user's password obfuscated (masked) by means of a pseudo-encryption algorithm:

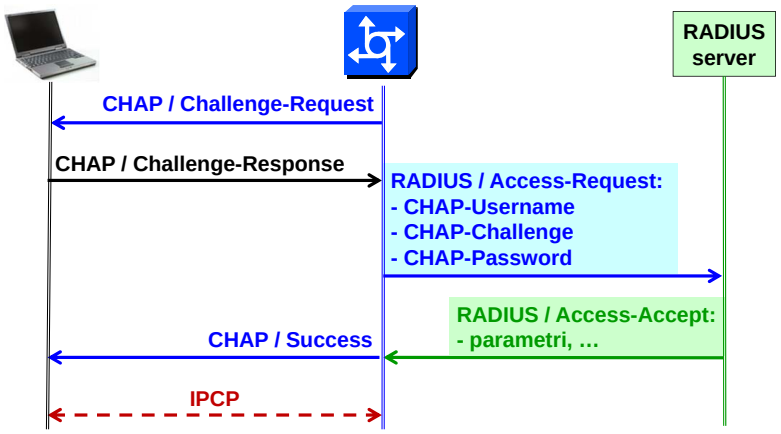
    (password || padding) ⊕ md5(key || Request Authenticator)

    * key: it is the secret shared with the AS ⇒ it provides NAS authentication against the AS;
    * Request Authenticator: it is random ⇒ two access requests by the same user will have different results;
  - Chap-Password attribute (type 3): it contains the user's reply to the challenge (128 bits);
  - Chap-Challenge attribute (type 60): it contains the challenge generated by the NAS to the user.

5.3.2 Example of authentication with CHAP

The user wants to connect through a NAS, which is connected to a RADIUS server:

1. CHAP / Challenge-Request: the NAS sends to the user a challenge;



2. CHAP / Challenge-Response: the user sends to the NAS the reply to the challenge, but just the RADIUS server has the user's password and can check whether the reply to the challenge is right;
3. RADIUS / Access-Request: the NAS sends to the RADIUS server an access request containing the following attributes:
   - CHAP-Username: the username provided by the user;
   - CHAP-Challenge: the challenge generated by the NAS to the user;
   - CHAP-Password: the reply to the challenge provided by the user;
4. RADIUS / Access-Accept: if the user is allowed and the reply to the challenge is right, the RADIUS server sends to the NAS the network configuration parameters for the user;
5. CHAP / Success: the NAS 'hands' to the user the network configuration parameters;
6. IPCP: it enables the IP connection.

5.3.3 Security analysis

Unfortunately RADIUS does not provide all security features which an authentication protocol would need in order to exchange NAS requests and AS responses in a secure way.

NAS request

- **sniffing**:
  - confidentiality: the password is not encrypted, but is just obfuscated:
    - − PAP: what is obfuscated is the cleartext password ⇒ the bad guy can steal the password, even though by very hard techniques;
    - \+ CHAP: the password is never transmitted ⇒ the bad guy can not steal the password;
    - \+ EAP: the password is not transmitted or is transmitted not in clear ⇒ the bad guy can not steal the password;
  - − privacy: the username is transmitted in clear ⇒ the bad guy can know when the user connected/disconnected;
- **brute-force attack** (password enumeration): the bad guy could discover the user's password by using the AS as an oracle:

    + authentication: the password is obfuscated also including the key which is the secret shared with the NAS ⇒ the bad guy can not pretend to be a NAS;
    + no reply: if the key is wrong, the request will be ignored with no error messages;
- **DoS attack**: the bad guy could saturate the AS with a lot of requests:
    + scalability: it allows to switch to a secondary AS if the AS is not available ⇒ saturating a single server is not enough.

AS response

- **faking**: the bad guy could allow the access to that invalid user or deny the access to that valid user:
    + authentication: the Response Authenticator is a MD5 keyed-digest: the key is the secret shared with the AS ⇒ the bad guy can not pretend to be the AS;
    + integrity: the Response Authenticator is a MD5 keyed-digest: the digest will not match if a change occurred ⇒ the bad guy can not change the AS response;
- **replay attack**: the bad guy could allow the access to another invalid user or deny the access to another valid user:
    + response tied to the request: the AS does not just reply by a simple yes or no, but the Response Authenticator is computed also including the Request Authenticator.

5.4 DIAMETER

The **DIAMETER** protocol is an evolution to RADIUS:

- it puts emphasis on roaming among different ISPs: users of an ISP can connect to another ISP in roaming (like in the cellular network);
- it cares about security more than RADIUS, because it is thought with security in mind:
    – RADIUS: security is implemented in the protocol, so new stronger security mechanisms can not be adopted;
    – DIAMETER: the main functionality of the protocol is detached from security, forcing to use an existing external security protocol.

However it is still not very used because:

- RADIUS works very well;
- roaming among ISPs has not been so considerable yet: two ISPs should carry out a commercial agreement to allow users roaming.

Compulsory security protocols The RFC specifies to compulsorily use higher-level security protocols:

- **IPsec** (layer 3: sect. 5.9): it must be supported by both DIAMETER clients and servers:
    – authentication and confidentiality: ESP must be used, with non-null algorithms, which protects the whole packet;
- **TLS** (layer 7: sect. 7.3.13): it must be supported by DIAMETER servers (and may be supported by DIAMETER clients):

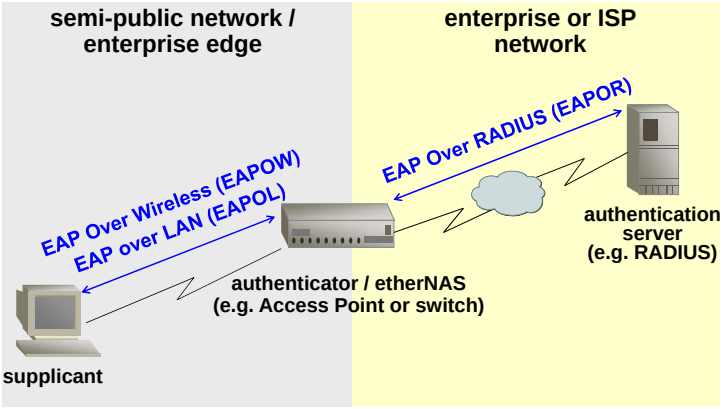
- authentication and confidentiality: the RSA + RC4_128/3DES + MD5/SHA1 combination must be supported (and the RSA + AES_128 + SHA1 combination may be used);
- mutual authentication: the server must authenticate itself too, and the client must have a public key certificate.

5.5 IEEE 802.1x

The **IEEE 802.1x** standard, named 'Port-Based Network Access Control', has been very successful since the first Microsoft (Windows XP) and Cisco implementations, and defines a generalized architecture which provides a framework to perform:

- **authentication at MAC level** (layer 2): the network access is restricted just to allowed users through an authentication asked by the network device itself:
  - wired network: the user is physically attached by a cable to a switch port ⇒ authentication is not indispensable because the user needs to have physical access to the port;
  - wireless network: the user connects remotely to an access point ⇒ authentication becomes indispensable because the user may be everywhere;
- **key management**: keys to protect communications (indispensable for wireless):
  - it can derive session keys to be used for packet authentication, integrity or confidentiality, depending on the security features chosen for communication;
  - it just provides packet formats to transport information, by exploiting standard algorithms for key derivation (e.g. TLS, SRP);
  - security services are optional: deciding if authentication only or authentication + encryption should be used is up to the client and the access point.

5.5.1 Architecture



From the architectural point of view three entities are distinguished:

- **supplicant**: it is the client of the user which is 'supplicating' to have network access;
- **authenticator** (or etherNAS): it is the access point to the enterprise or ISP network, that is the switch or the wireless access point wireless, which is in charge of:
  - blocking the network access until the client is authenticated successfully;

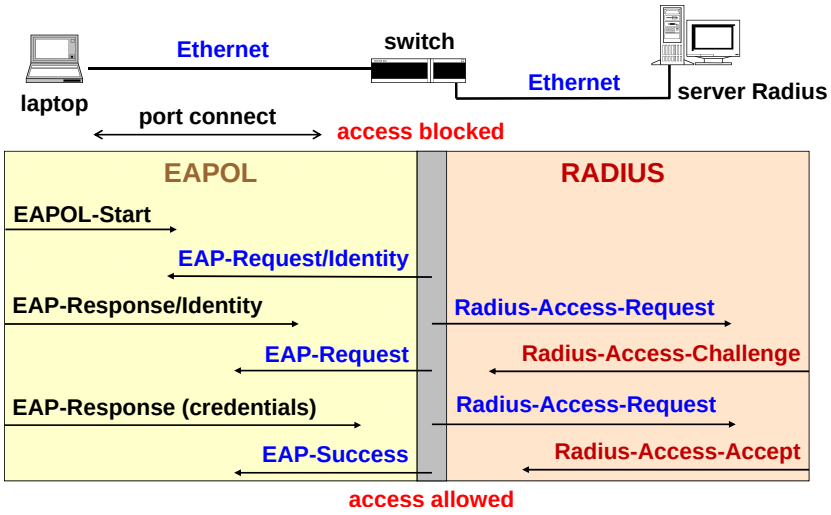
  - acting as a proxy toward the server during the authentication step;

- **authentication server**: it is the RADIUS AS, which is in charge of accepting or rejecting access requests.

The supplicant's access request goes to the authentication server through the authenticator:

1. the supplicant sends to the authenticator an access request EAP message through the semi-public or enterprise layer-2 network:
   - EAP Over LAN (EAPOL) if the network is wired;
   - EAP Over Wireless (EAPOW) if the network is wireless;
2. the authenticator encapsulates the EAP message into RADIUS (EAP Over RADIUS [EAPOR]) and sends it to the AS through the enterprise or ISP IP network.

5.5.2 Messages



When the user connects his laptop to the port of an Ethernet switch:

1. port connect: the switch detects there is electrical activity on the access port, which is in default status 'blocked' it can exit only at the end of an authentication process;
2. EAPOL-Start: the 802.1x software on the client asks the authentication process to be started so as to get access to the network;
3. EAP-Request/Identity: the switch asks the client to show its own identity;
4. EAP-Response/Identity: the client sends the requested information about its own identity;
5. Radius-Access-Request: the switch forwards this information to the server, re-encapsulating it into the RADIUS data format;
6. Radius-Access-Challenge: the server generates a challenge for the client;
7. EAP-Request: the switch forwards the challenge to the client, re-encapsulating it into the EAP data format;
8. EAP-Response: the client provides its credentials, containing the username and the reply to the challenge;

9. Radius-Access-Request: the switch forwards these credentials to the server, re-encapsulating in the RADIUS data format;
10. Radius-Access-Accept: if the reply to the challenge is right and the user is allowed to access the network, the server accepts the access request;
11. EAP-Success: the switch forwards this reply to the client, re-incapsulating it into the EAP data format;
12. the switch unblock the port, and gives to the client the network configuration via DHCP.

Unlike CHAP authentication, the challenge is not generated by the NAS but by the RADIUS server.

5.5.3 Advantages

802.1x does not invent new protocols, but uses already existing protocols:

- direct conversation (at conceptual level) between the supplicant and the AS: the supplicant's network card (NIC) and the authenticator only act as 'pass-through' devices;
- it exploits the EAP protocol for the actual implementation of security mechanisms:
  - all EAP-compatible authentication systems can be used;
  - easily extensible: support to new authentication methods does not require changes to neither the NIC or the authenticator;
- it exploits the RADIUS protocol being seamlessly integrated with the three As:
  - accounting: the company or the ISP can log when each user is connected or disconnected ⇒ it is useful to know who was connected to the network while an attack was happening (audit).

5.6 DHCP security

DHCP is a protocol which lies between layer 2 and layer 3, and is used to provide the network configuration to a node which is without it and wants to enter an IP network.

DHCP is a **non-authenticated protocol**: besides the fact that the user asking for the network configuration is not authenticated, the main problem is that the reply is not authenticated ⇒ it is very easy to set up shadow servers as fake DHCP servers, making possible two kinds of attacks:

- **DoS attacks**: the client is provided with a wrong and not working network configuration (e.g. wrong DNS address) which prevents him from surfing ⇒ the user realizes that and calls the network administrator;
- **logical man-in-the-middle attacks**: the client is provided with a wrong but working network configuration ⇒ the user does not realize that:
  - outgoing packets: the client belongs to a network with netmask "/30", and its default gateway is the shadow server;
  - incoming packets: replies can be intercepted by defining a NAT.

There are some attempts to provide protection to DHCP:

- **DHCP snooping**: the administrator defines at which switch ports replies by DHCP servers can arrive ⇒ an attacker can not send DHCP replies by connecting to a non-enabled port;

- **IP guard**: packets with source IP address not included among the ones provided via DHCP are dropped ⇒ it limits IP spoofing attacks, but the switch should store a lot of IP addresses;
- **authentication for DHCP messages**: DHCP replies are authenticated by a MD5 HMAC keyed-digest ⇒ it would be the ideal solution, but:
  - it is rarely adopted;
  - distinct keys should be used, otherwise a bad guy just by having a valid client will know the symmetric key and use it for a fake DHCP server.

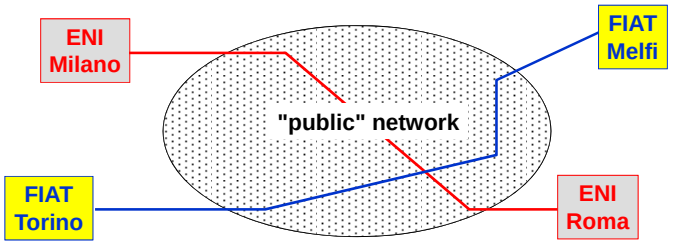
5.7 Security at network layer

The network layer is the first layer being really interesting in the OSI stack where to implement security features, because it is the first layer to connect computing nodes in an end-to-end way.

At network layer two kinds of security features can be created:

- end-to-end protection: the client directly negotiates with the server the security features:
  - \+ the protection works even if attacks occur along the tunnel;
  - − it requires configuration of all clients, but users may not be so skilled;
- network protection (e.g. VPN): security features are on intermediate nodes (routers):
  - − the client and the server are the weak points in the network;
  - \+ protection is transparent to users.

5.8 VPN



Virtual Private Network (VPN) a hardware and/or software technique to create a private network while using shared or anyway untrusted channels and transmission devices

For example, suppose that there is a public network (or equivalent in public, that is, not of personal use) on which you want to convey their traffic. A company that uses a network of this kind, and who wants to be able to communicate securely with all its offices, will somehow mark packets within its competence; for example, in the figure the company ENI has marked its packages in red, while the Fiat has marked their packages in blue.

In such a situation you have a number of problems:

- How can you be sure that the packages labeled in a certain way not been tampered with? A package "red" is really that color or has been changed?
- How to protect yourself against those who manage the public network?

There are several techniques for the realization of a VPN:

- Using directives hidden.
- Using secure routing (IP tunnel).
- Use of secure routing and safe (secure IP tunnel).

5.8.1 VPN network using hidden

This type of VPN is based on the use of a non-standard address so as not to be accessible from other networks. Example of this technique is the use of hidden networks IANA according to the RFC-1918.

From a security perspective this system is extremely weak because:

- If someone discovers the addresses used and decides to change the address and get into another class it would be able to have access another network.
- Who has control of the infrastructure can read and manipulate the packets in transit.

5.8.2 via VPN tunnel

With this technique, the router to ensure encapsulate network packets within other packages, adopting the routes so that the different branches of a company can only communicate with each other. The encapsulation techniques may be different:

- If you want to stay at the IP layer can perform an IP in IP encapsulation.
- IP over MPLS.
- Other implementations.

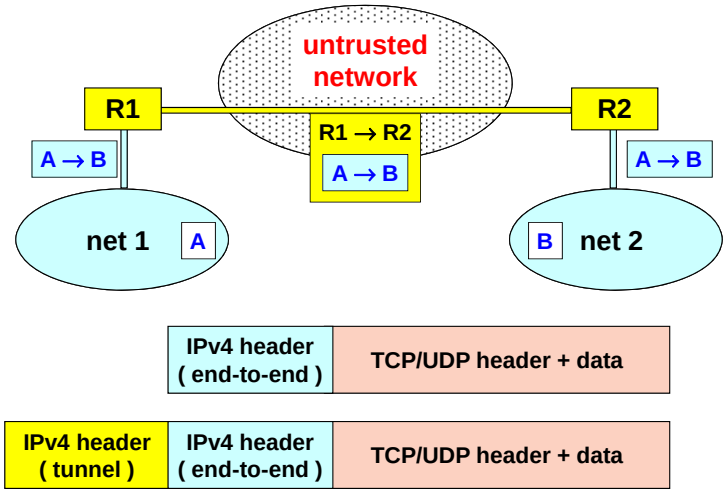
The important thing however, regardless of the type of deployment that you choose is not that you have to use to route the original addresses, but you must create a new addressing constraints that packets to be exchanged only between sites belonging to the same VPN logic.

Again the router control access to networks using the ACL (Access Control List).

This technique is used mainly to the supplier of telecommunications services to protect against users who try to gain access to different networks from their home network. It therefore has a minimum of safety in the case where a user tries to enter a class in another class.

However you do not have any protection for users of VPN service towards who manages the network. This is because those who have control of the network can always go to manipulate all packages at will.

VPN tunnel using IP



In the graph you can see two IP networks (blue), a node A that wants to communicate with a node B and between the two nodes a public network.

The original package sent from network1 has a header end-to-end in which the sender appears to be the node A and node B is the recipient instead; this header is followed by the corresponding payload.

The moment this packet arrives at the border router R1 public network it will see that the package has as its destination the network managed by the router R2. As a consequence the packet will be encapsulated in a new packet, in which the header of the IP layer will be from the router R1 and the router R2 as a recipient. In other words, the original packet becomes the payload of a new package, to which is appended a new header that connects the end-to-end point of the tunnel.

As a result of this operation, the encapsulated packet travels as in a gallery in the public network, and after coming to the router R2 this will remove the header of the tunnel going to restore the original package, and finally going to deliver it to the destination node .

As can be seen from this example, those who manage the public network is free to manipulate data at will, whereas network operators 1 and 2 will not try to have access to other networks because it will be the border router to force the' routing.

Tunnel IP: fragmentation

Make the IP tunnel has a cost, in the sense that it is based sull'incapsulare an IP packet inside another package and this can lead to generate a packet with a size greater than MTU. If this occurs must resort to fragmentation, which leads to a loss of performance of up to 50 %.

The performance loss real still depends on the size of the packets. In general, applications that suffer most are the bulk transfer applications, ie those not interactive.

5.8.3 VPN tunnel through secure IP

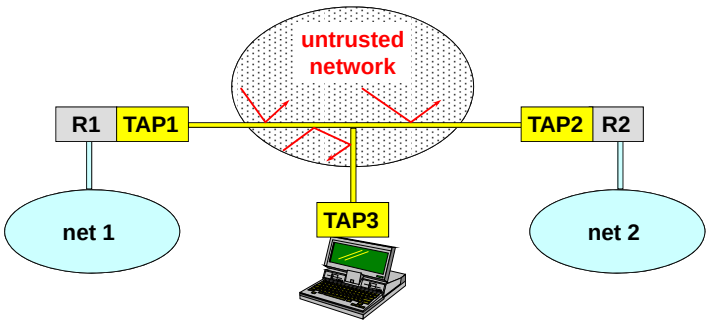
This type of VPN is always based on the use of a tunnel, but the packets before being transferred are protected. This is done mainly to protect themselves against the operator of the public network. Packages can be protected with:

- Digest, to protect integrity and authentication. This prevents the network operator can modify or even add packages.
- encryption for privacy, so that you can not read the packages.
- numbering to prevent replay attacks.

The important thing about this technique is that the protection must be made directly from the customer's VPN, and not by the operator. Also, if the customer decides to use of strong cryptographic algorithms, then the only possible attack is to prevent communication using DoS.

This technique of VPN is also called S-VPN, ie Secure VPN.

Example



In this example, in addition to be the routers R1 and R2 that deal with the encapsulation process you have another node in which are implemented additional security features that the customer wants to have.

The arrows within the public network are to indicate that attacks "bounce", have no effect.

5.9 IPsec

IPsec is the architecture proposed by the IETF to make security level 3 for both IPv4 and IPv6. IPsec allows you to:

- Create VPN on an untrusted network.
- Making end-to-end between two nodes of an IP network.

To implement its functionality IPsec amending Protocol IP going to add two special headers:

- *Authentication Header (AH)* which provides additional features such as integrity (package), authentication (the sender) and no replay.
- *Encapsulating Security Payload (ESP)* that provides all the functionality of the header AH more functionality to have privacy.

Looking at the features offered you can guess as they are based on the use of cryptographic keys that must be negotiated between the nodes of the IP network. It uses about it a protocol for key exchange called *Internet Key Exchange (IKE)*, which is used to negotiate the algorithms and keys to be used to implement the security features of AH and ESP.

Summing IPsec provides authentication of IP packets:

- Data Integrity transported.
- Identification of the sender.
- Protection (partial) from replay attacks.

The confidentiality of IP packets is instead offered through data encryption.

5.9.1 IPsec Security Association (SA)

All the features of IPsec are made by the concept of Security Association, ie two nodes of an IP network to qualify for the IPsec protocol must first negotiate among themselves a security association (something who decides what security features must have the traffic between the two of them).

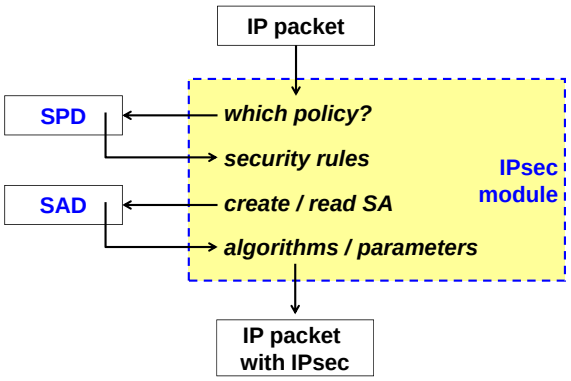
SA is a unidirectional association, and therefore if you want to protect traffic in both directions between two nodes will have to go to create two SA.

You have no obligation to give the same safety features to both SA.

To manage the SA and overall IPsec traffic type, each node that implements IPsec must also implement two DB (DB from logical point of view):

- *Security Policy Database (SDP)* the which contains the security policy to be applied to different types of communication. It can be configured in advance (eg. Manual) or it can also be coupled to automatic systems (eg. ISPS - Internet Security Policy System).
  The policy provides information about security features that must have a certain type of traffic.
- *SA Database (SAD)* that contains the list of active SA and their characteristics (algorithms, keys, parameters).

5.9.2 How IPsec (shipping)



Suppose you observe an outgoing IP packet from a network node.

If you have enabled IPsec protection for IP traffic, the packet IP when it comes out of the level 3 of the OSI stack, before being sent to Level 2 is intercepted by the IPsec module. The packet is inspected and the module, after making a lookup within the SPD, determined according to the characteristics of the package if you need to or not to apply some security policies.

If the package does not require any policy then it will go directly to the second level of the stack, and if so the form will go to SA to assign a specific package. If this is the first packet that is exchanged between the two nodes of the network then you must create a SA and record within the SAD; but if this is not the first packet in the flow of the two nodes, but it is a package that corresponds to an SA already registered then you will need to do a lookup in the SAD in order to retrieve the data associated with the SA, and so go to apply them to the package.

The result of these operations will be increased by an IP packet with IPsec. The level 2 will then have the task of transmitting to the destination.

5.9.3 Where to apply the functionality of IPsec?

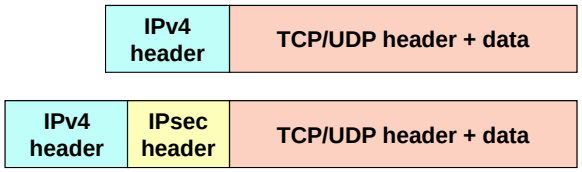
When you go to protect a package you have to be very careful about which components of the package can be protected.

Regarding the functionality of confidentiality typically applies only to the payload because if you go to even encrypt the IP header intermediate routers would not provide some necessary information for routing decisions.

Regarding instead authentication and integrity at first glance you might think that it is applied to the entire package. However this is not true, as some within the header fields are changed gradually that the packet passes through the various router communication. As a result of these two security features surely will cover the payload, but also the fixed fields within the IP header.

5.9.4 IPsec in transport mode

This mode of use of IPsec is to take the original package and insert between the IP header and the payload header IPsec additional.



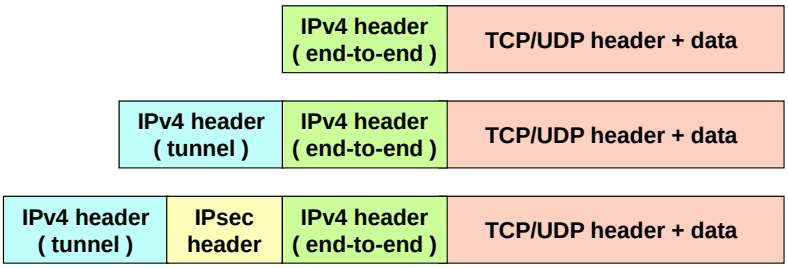
By performing this operation must also go to update the protocol field in the IP header, so it appears that the IP header is used to transport an IPsec header. It is as if IPsec was considered as a pseudo-protocol level 4, but actually inside the header IPsec you an indication of the level 4 protocol actually used.

IPsec transport mode is usually used to make end-to-end, that is used by hosts and gateways. The exception is when the intermediate nodes behave themselves from end-node, for example when using SNMP or ICMP.

The advantage of this mode is that the steps are few, and so it is computationally light. The disadvantage is that you do not provide any protection of IP header variable fields.

5.9.5 IPsec in tunnel mode

This mode of use is the one that allows you to create the VPN, and therefore is generally used by the gateway.



The basic idea is to take the original packet (header + payload), go to encapsulate it within another IP packet and the resulting package is going to apply IPsec protection. With this technique the IPsec header goes to protect the entire contents of the original packet, including its header; the only parts that are not protected are the variable parts of the tunnel.

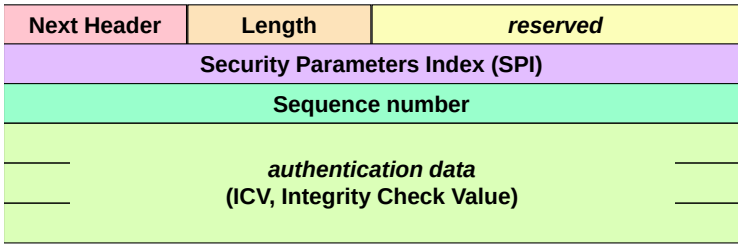
The advantage of this technique is the protection of the original package, including the variable fields (which are no longer variables, remain fixed inside the tunnel). The disadvantage is that you have a greater computational cost compared to the packages in transport mode, because you must first create a tunnel and then go to protect it.

5.9.6 Authentication Header (AH)

were implemented two versions of this header:

- The first version (RFC-1826) allowed to have only data integrity and sender authentication. How keyed digest were foreseen algorithms keyed-MD5 (required) and the keyed-SHA-1 (optional).
- The second version (RFC-2402), which is now the most commonly used, as well as integrity and authentication added functionality no replay. The algorithms used by this version are the HMAC-SHA-MD5-96 and HMAC-1-96.

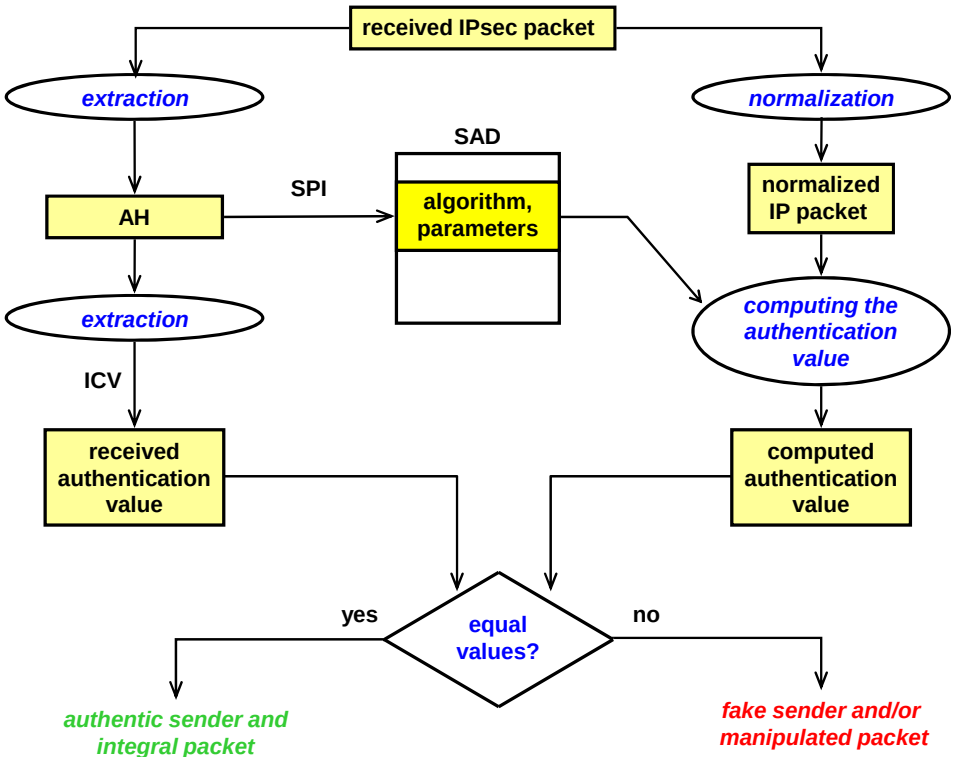
AH - Format (RFC-2402)



This header resides between the IP header and payload, and contains all the information needed to authenticate and to ensure the integrity of the package.

- Next Header (1 byte): preserves information level 4 protocol original.
- Length (1 byte): length.
- *Reserved* (2 bytes): bits reserved for future use.
- *Security Parameter Index, SPI* (4 bytes): SAD index in the table. This identifier is used to associate the header to a particular SA. To find what are the algorithms and in general the parameters applied to the package containing an AH header, the sender and the receiver must perform a lookup in their respective tables SAD using a triplet: address _mittente / address _destinatario / SPI.
- Sequence Number (16 bytes): field used to number the IP packets belonging to a single SA. Is to prevent replay attacks.
- Authentication Data (12 bytes): These bits are the so-called ICV (Integrity Check Value). Data needed to provide authentication and integrity. It is the sender to calculate these data; the recipient has the task to recalculate the received packet and going to compare them.

AH - Procedure



You receive an IPsec packet, protected with AH, and want to know if it is intact and authentic.

The first operation that is performed is to go to extract from the packet data relating to AH. From these data we can derive the ICV, and in this way it is possible to know the value of authentication that the sender has calculated and included within the package.

Without this first step you have to check if this authentication value corresponds to the data received. It then makes a normalization of the IPsec packet (in short you put in the same

condition in which the calculation was made by the sender). Once you have available the IP packet is necessary to calculate the normalized keyed-digest; But to do this you need to know the algorithm and key, which are obtainable by performing a lookup in the SAD-based SPI contained within the AH.

You are now able to calculate the authentication value on the received data. If the authentication value received is equal to that calculated, then the sender is authentic and the package is intact, while otherwise the sender is false and / or the package has been tampered with. In the latter case the packet will be discarded.

If the operation had a positive result, those who have been authenticated? The only certainty that we have is that the package was sent by the one who negotiated with the recipient of the SA. Who is this one depends on the type of authentication used in the moment of creation of the SA.

AH - Normalization

To Normalize is to put yourself in the same conditions of the sender, which is reset the TTL field (the sender put a certain value, while the recipient would receive equal to the initial value minus the number of routers that have been crossed). This operation is carried out by both the sender and the recipient, but only for the purposes of calculating the value of authentication.

In the event that the package contained a Routing Header must then:

- Fix the destination field at the recipient's end.
- Attach the content of the routing header to the value that will have a destination.
- Secure Address Index field to the value that will have a destination.

Any other option that changes during transit (ie presenting the bit C, Change En Route) in the network must also be reset.

AH - HMAC-SHA1-96

Given a package M is done by generating the normalization package M'. The operation of this algorithm is the following:

- You make a 160-bit alignment of M' (adding bits to zero), so going to generate $M'p$.
- The key K is also aligned 160-bit (adding bits to zero), so as to create the key K_p.
- Given the fixed values ip = 00,110,110 and op = 01011010 (both repeated so as to form 160 bits) you can now calculate the basic authentication by following the formula

$$B = \text{sha1}\left((K_p \oplus \text{op}) \,||\, \text{sha1}\left((K_p \oplus \text{ip}) \,||\, M'p\right)\right)$$

- Because the size of B is variable depending on the algorithm used, which would generate IPsec packets of size not fixed, then we have that

$$\text{ICV} = 96 \text{ leftmost bit di } b$$

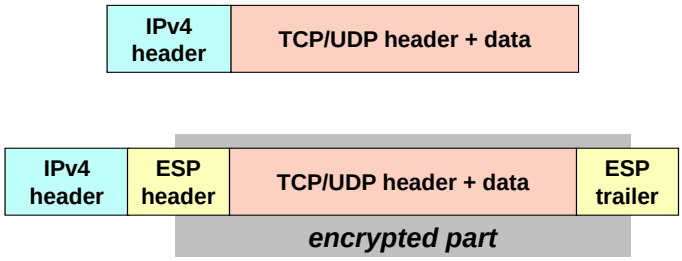
5.9.7 Encapsulating Security Payload (ESP)

In its first version (RFC-1827) ESP packets were orthogonal to the packages AH, ie provided only confidentiality. If you then want to have authentication, integrity, confidentiality and no replay of the same package was necessary to go to implement both AH is ESP.

In the next version (RFC-2406) has decided to implement in ESP also most of the functionality of AH, in particular part of authentication, integrity and no replay. It was only one exception: ESP package does not include the IP header, and then the security features unique to the payload of a packet. The advantage is the reduction of the size of the package.

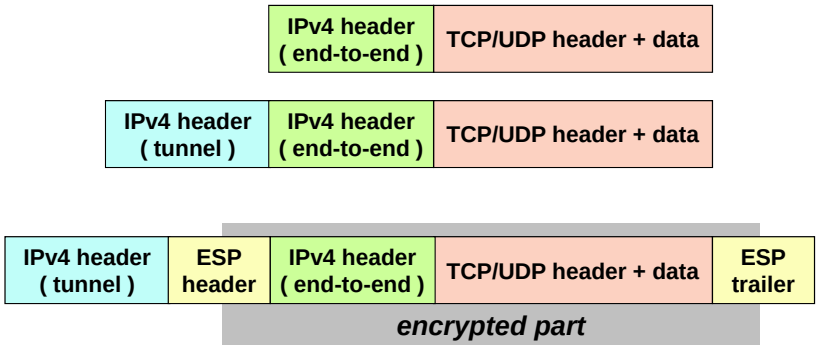
ESP in transport mode

ESP in transport mode the encrypted part includes only the payload; the original header remains completely in the clear.



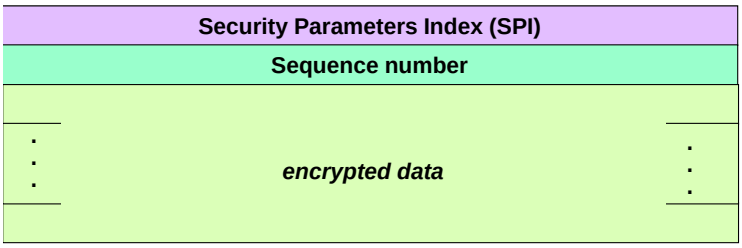
ESP in tunnel mode

If ESP is instead applied in tunnel mode, as the payload is to be all original package are therefore also hidden the true sender and recipient of the package.



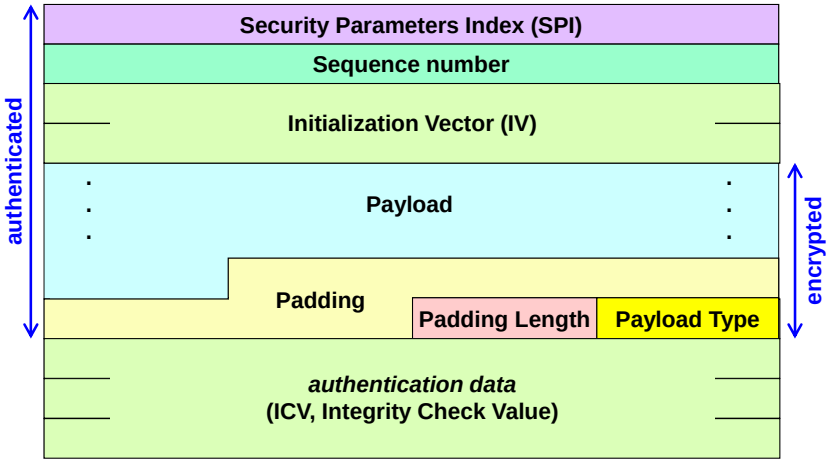
What remains clear is the header of the tunnel. Any actant then you will not be able to know who is using the tunnel to communicate.

ESP - Format (RFC _2406)



- Security Parameter Index (SPI): indication of what SA refers to this package.
- *Sequence Number*.
- Encrypted Data: To read this data you must be aware of the SA used to create them.

Assuming that the data is encrypted using the DES-CBC, the packet format is as follows:



After the sequence number are the 128-bit initialization vector (*Initialization Vector, IV*), field used by the algorithm CBC; in this field follows the *payload*. Since the DES requires that the payload is a multiple of 64 bits, it will be followed by the Padding: the size of the added padding will be indicated in the field *Padding Length*, which will be followed by the field *Payload Type* ie what is the higher-level protocol that is transported.

Fields payload, Padding Length and Payload Type is the data that is encrypted, while data are authenticated these fields in addition to the initialization vector, the Sequence Number and SPI.

The authentication data (96 bits) are placed at the bottom of the pack. The ICV is calculated in exactly the same way as it was calculated for AH, with the exception that not having to go to treat the IP header is not necessary normalization procedure because in this case the data are all fixed.

5.9.8 Implementation details

Because you can go to choose among several algorithms, defined two crypto-suite (set of algorithms) RFC-4308 in order to facilitate interoperability when you realize VPNs:

- VPN-A: involves the use of ESP packets with 3DES-CBC encryption, and authentication and integrity given by HMAC-SHA1-96. This represents the minimum level of security that you can rely on.
- VPN-B: is always based on the use of ESP packets but with AES-128-CBC, and the portion of the authentication data by AES-XCBC _MAC -96. This solution, in addition to offering a higher security level compared to the previous solution, has the advantage of using ES both for the integrity of both for the authentication part, and therefore it is not necessary to go to implement additional algorithms.

Very used are also algorithms NULL, both for the authentication for that encryption. This allows you to use a single format (such as ESP) and modulate what you want to go to implement. For example, if you wanted to have ESP only for privacy, you can go and specify NULL for the part of keyed-digest and therefore you will only confidentiality.

In general, therefore these algorithms are used to have a trade-off between security and performance.

The last implementation detail but very important is that concerning the Sequence Number. It provides protection from replay, but only in a partial way; this because it is associated with a window that covers what packages are covered by the protection from replay. The standard requires at a minimum that should be considered the last 32 packets received, but the advice is to consider the last 64 to have better protection.

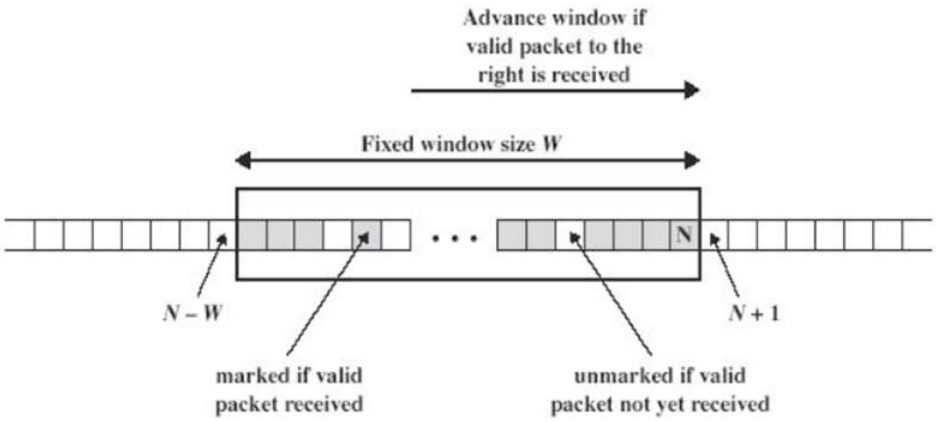
5.9.9 replay protection in IPsec

First, it should be noted that IPsec is applied to IP packets. The IP network but is not reliable and the packets may be lost; consequently IPsec can not provide protection in any way on packet filtering and packet cancellation. The replay protection is only partial because IP has the problem of being a non-sequential protocol, ie the order in which packets are received may not correspond with the order in which packets have been transmitted. This means that when you go to check a replay attack is not possible to go to see only the sequence number, as a packet with a sequence number less might arrive at the destination after a packet with sequence number greater. To know if a package has already been received or not so do not just keep the sequence number of the packet with the highest number received, but you need to keep track of all packages that are received. But this is impossible, because it would need to have a huge memory to keep track of all packages. Hence the concept of the window, that you have a buffer in which it keeps track of the last sequence numbers of the last N packets received.

This concept is implemented sender side as follows:

- When you create a SA sender initializes the sequence number to 0.
- When you send a package increases the Sequence Number.
- When you reach the Sequence Number maximum it negotiates a new SA (otherwise it would overflow resulting in automatic replay attack).

The recipient instead uses a window of fixed size W.



The packages in gray are those that have already been received, while in white are the packages that have yet to be received.

When you receive a package to the right of the window, it moves right up to include it. In this way the packages are excluded older.

Where there is a replay of a package that is inside the window, it is identified immediately. If the replay has as its object a packet outside of the window, there is no way of knowing whether it is or not replay. In the latter case it is possible to have two strategies:

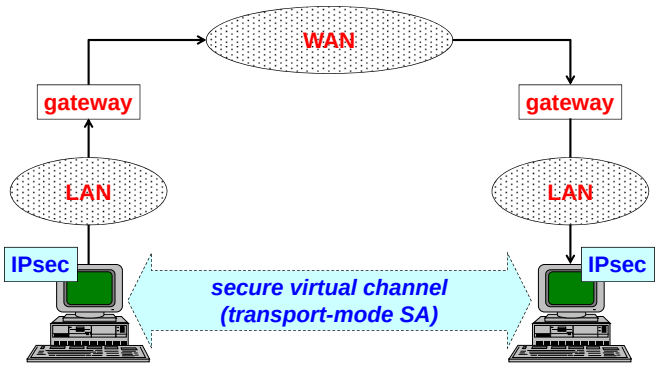
- The packet is accepted, however, and is sent to the upper levels. If the upper level is TCP then it will be to manage the package in the most appropriate way (going to accept it if it were really a missing package, or going to reject it if it is a package that has already been received), but in the case where UDP had the situation is less manageable since UDP does not have the concept of the segment, and then the package may be accepted even if it has already been received earlier. UDP attacks therefore are much more dangerous for UDP than TCP.
- If the package is too old, and then out the window, the IPsec stack it goes to discard anyway. This solution provides better protection from replay, but it may cause slowdowns as it may cause retransmissions.

Summarizing the protection from replay attacks depends on the size of the window and the strategy used for the management of received packets outside the window.

5.9.10 Architectures protection

For a system to be declared IPsec-compliant must be able to implement at least four basic architectures.

End-to-end security



This architecture takes this name because it is one in which IPsec is used to secure traffic between two nodes directly client-server.

This means that the two nodes on which you installed the software IPsec negotiate among themselves a mode SA transport mode. In other words the two nodes you create a safe virtual channel, with safety features that have negotiated, which makes the protection of the traffic generated by them independent of everything: it is not important if the LAN are not safe, if the gateways are managed by people who do not trust or implements or manages WAN. If the two machines are safe then you will have the full protection of network traffic, with the exception of DoS.

This architecture has both advantages and disadvantages of:

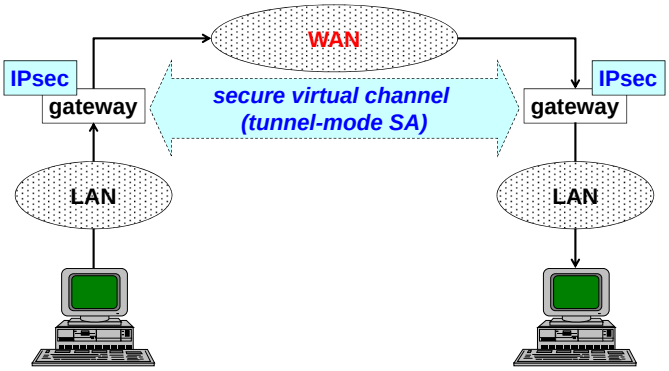
+ The two nodes are rendered completely independent and protected from the rest of the network, both local and geographical.

– is necessary go to install IPsec on all nodes of the network that you want to go to protect; consequently this solution is mainly used when you have a few nodes to be protected.

– The fact of going to install IPsec on single node assumes that node has the cryptographic capabilities necessary; this is very important as far as the client, but also with regard to the server in the case where the number of nodes was high.

– In the case that the channel has been implemented confidentiality manage the local network can not go traffic monitoring.

Basic VPN



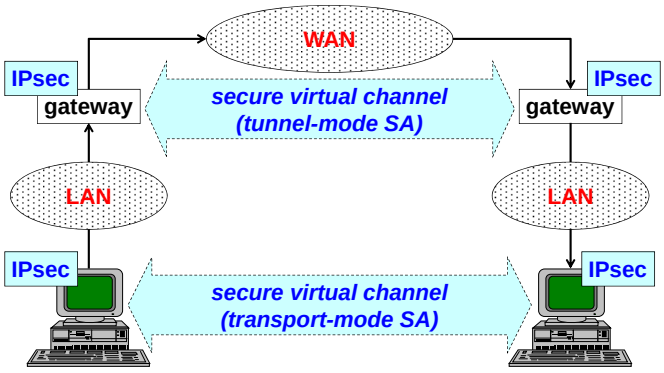
This architecture implements the opposite idea than previous architecture: instead of going to implement IPsec on end-node, it is assumed that local networks are trustworthy and which ones are dangerous only the wide area networks. Based on this idea IPsec is installed on the gateway, ie access points between two networks with different levels of security: a secure network you trust (in the image was defined as LAN) and one of which is not trusts (WAN).

Advantages and disadvantages:

+ No more going to manage a large number of nodes, but simply managing the only gateway.
+ If the gateway had of computational problems is possibile andare a prendere delle misure andando ad installare degli acceleratori crittografici.
+ La parte di rete fidata non essendo più cifrata può essere soggetta a monitoraggio.
− Non si ha più una protezione end-to- end. This means that if there is any attacker within the trusted network that is free to perform any type of attack it wants.

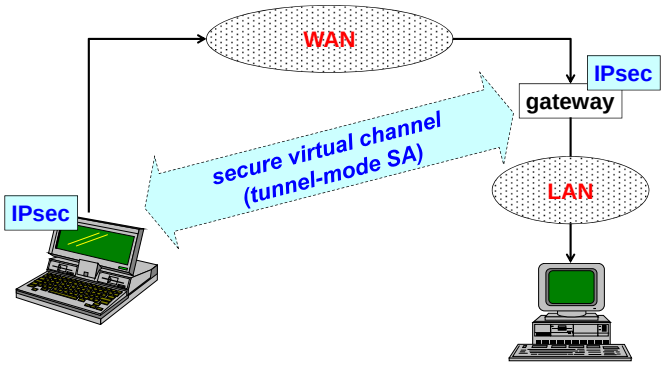
Today this is one of the most used.

End-to-end security with basic VPN



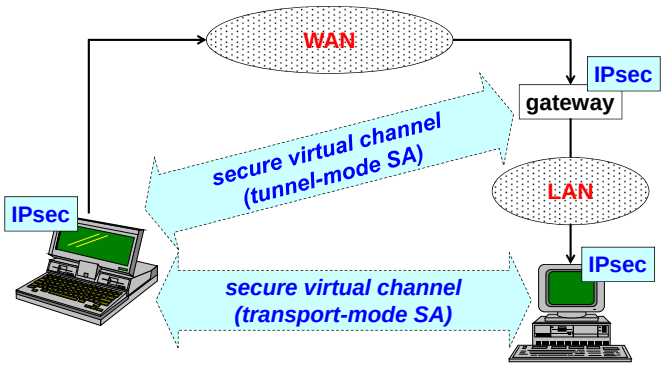
This architecture implements a double line of defense, going to install modules on both IPsec end-node is the gateway. This can be done either to go to double the level of defense, both to go to divide the defenses: for example on the virtual channel transport-safe mode could be implemented but only with IPsec AH, which would ensure the authentication and integrity of the data and allow to manage the network traffic monitoring, while the part of confidentiality is implemented on the gateway assuming that the most dangerous part as regards both the part of sniffing untrusted network.

Secure gateway



Today much importance is acquiring the secure gateway architecture, in which a mobile node is equipped with software IPsec and goes to realize a safe virtual channel in tunnel-mode with an access point to a network. With this solution not only implements confidentiality but also authentication (usually at the application level, so as to implement a system of management of access).

Secure remote access



In this fifth architecture not only is it a safe virtual channel in tunnel-mode to the gateway, but it also has a safe virtual channel in transport mode towards the end node. This is done to double security features, or to divide. in the latter case the authentication is typically placed at the level of the tunnel-mode because you want to keep track of who is trying to access the network from outside.

5.9.11 IPsec key management

A key component of IPsec is the part of key management, which provides systems that communicate symmetric keys for authentication and / or encryption packages. The problem is going to decide how to go and distribute these keys; this can be done:

- manually, going to configure the keys directly on the nodes that communicate with each other.
- In automatically.

ISAKMP

Internet Security Association and Key Management Protocol is the protocol used to negotiate keys and establish the SA.

It defines all the procedures that are used to negotiate, establish, modify and delete SA. This protocol is a framework, in the sense that it is used to transport packets but does not indicate the method to be used for the key exchange. Currently with ISAKMP is very often used protocol OAKLEY, which realizes the exchange authenticated symmetric keys between systems IPsec.

IKE

Internet Key Exchange is nothing but the combination between ISAKMP and Oakley. It has been standardized as it is a very useful solution.

Protocol particularly complex as it requires you to create a first SA not to protect the exchange between two nodes but to protect the exchange ISAKMP. This SA is protected negotiation of subsequent requests from IPsec SA. The same ISAKMP SA can be used several times to negotiate other IPsec SA, always between the same nodes.

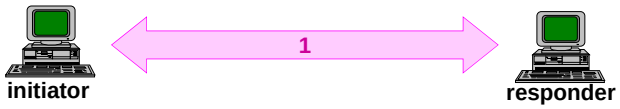


Figure 5.2: IKE phase 1 - ISAKMP SA negotiation of a two-way: "main mode" or "aggressive mode"

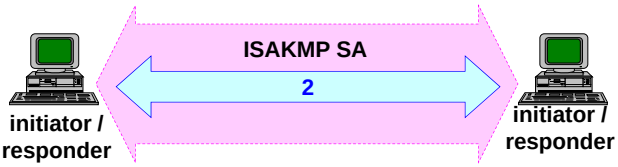


Figure 5.3: IKE phase 2 - negotiation of IPsec SA: "quick mode"

From a functional point of view there are two nodes wishing to communicate, and one of the two takes the initiative; For this reason, this node is called initiator. This node Call responder and must implement the IKE phase 1, which is negotiated an ISAKMP SA bidirectional; this can be done in two ways, *main mode* or *aggressive mode*.

Created this first SA, once one of the two nodes want to open a communication channel to the other node will create a second SA but for the use of IPsec. This represents the IKE phase 2 which, thanks to the fact that is performed within another SA can be made so much faster, and for this reason it is called *quick mode*.

The different modes of IKE have the following features:

- Main Mode: is the heaviest of all because it requires the exchange of cryptographic 6 posts. The advantage is to protect the identity of the parties.
- Aggressive mode: so called because it requires far fewer messages than the previous mode; in particular three packages are sufficient cryptographic has the advantage of being extremely fast but does not protect the identity of the parties.
- Quick mode: This part uses only 3 posts, and is only used for trading of the IPsec SA .
- New Group Mode: given an existing SA, this mode is only used to change the cryptographic parameters. It is based on the use of two messages.

As for the authentication part, when you create the SA can do:

- Digital Signature. Because it has the support of X.509 certificates is possible to have non-repudiation of the IKE negotiation. In other words, a user can not deny having created a certain SA.
- Public Key Encryption: authentication method is not based on X.509 certificates but on methods of public key cryptography, in order to protect the identity of the parties in'aggressive mode.
- Revised Public Key Encryption: compared to the previous solution this turns out to be less expensive because it uses only two public key operations.
- Pre-Shared Key: authentication mode easier because the keys are not traded, but is negotiations which keys will be used. This means that the nodes will have within them the keys, and when exchanging ISAKMP is only decided which key to use to go. In this case the constraint is that the identity of the counterparty may be only its IP address; as a result there is a problem for mobile users, who can not use the Pre-Shared Key as their IP address is variable.

5.9.12 IPsec - System Requirements

Buying a VPN concentrator is not essential. It is possible to go to implement IPsec on other nodes, for example:

- Router: You must have a router with a powerful CPU, or you need to go to give the router a cryptographic accelerator. In any case it is essential that the router is not outsourced, because otherwise you would entrust to others not only network management but also security.
- Firewall: being also a point of a network boundary between safe and unsafe you can go there to install the IPsec gateway. You again need to have a very powerful CPU.
- VPN concentrator: To improve performance when using extensively the different modes VPN often are bought directly hardware devices already dedicated to these purposes. These VPN named VPN concentrator. They are of special-purpose equipment which act as terminators of the IPsec tunnel, both in the case of remote access of individual client is to create the VPN site-to-site.
  The big advantage of these machines is that being custom machines not only have very high performance but above all they have the very low costs. In addition, they offer maximum independence from other security measures, allowing a segmentation of multiple layers of security measures.

5.9.13 Influence of IPsec performance

IPsec has an influence on performance. In particular implement IPsec means decreasing the throughput of the network; This is because:

- Maggiore packet size: In the carry mode AH will be added at least 24 bytes, while the carry-mode ESP DES-CBC has a minimum increase of 32 bytes; in the latter case the increase depends on how much padding is added.
- Greatest amount of packages: We must remember the packages needed to enable the SA. As a result there will be higher latency times, in the sense that the time set-up of the connections will be slower.

In general the decrease of the throughput is low, but this is not always true. For example the implementation of IPsec cause large delays if you go to replace a point-to-point link with a physical point-to-point virtual link; This is because usually the physical connection you make a

2-level compression of the data, but this would become useless or even harmful if associated with ESP packets.

To try to remedy the increase of packages you can compress packets before sending them to the IPsec module. This can be done via the protocol IPComp, which performs a compression of the payload to the IP layer, or the compression can be done directly at the application level.

5.9.14 Applicability of IPsec

- First seen that IPsec requires to negotiate an SA explicit with a certain node, you can go to apply exclusively to unicast packets. IPsec can not therefore be applied to broadcast packets, multicast or anycast. More generally IPsec can not be applied to all the packages in which you can not identify with certainty the recipient as it will not be possible to have a SA.

- Unicast packets can be protected only if it is possible to identify with certainty the counterparty via a SA. This means sharing with the counterparty a key in OOB mode, or the other party has been identified by X.509 certificate.
  This implies that in general IPsec is applied to groups "closed": closed because it has had a distribution OOB keys, or because they are virtually closed groups that trust the same CA.

5.10 Security IP

IPsec definitely protects the traffic of higher layer protocols, but does not completely solve the security problems at the IP layer.

IP generally used to transport not only TCP and UDP protocols, but also many others. Unfortunately, IP is a protocol to be very insecure because:

- The addresses are not authenticated.

- The packages are not protected, and then you do not have integrity, authentication, confidentiality or replay.

This implies that they are attacked all protocols that use IP as the transport, especially those of "service" that is, those of the application layer as ICMP, IGMP, DNS, RIP, etc.

5.10.1 Security ICMP

The Protocol and Internet Control Management Protocol (ICMP) is vital for network management. Being deprived of protection you can have many types of attacks, in particular:

- An attacker could exploit packet destination unreachable (used by an intermediate node to communicate the impossibility to deliver a package to the recipient) so to block traffic to a destination node, causing a DoS. This is because the receipt of a packet destination unreachable the sender node stops transmitting packets.

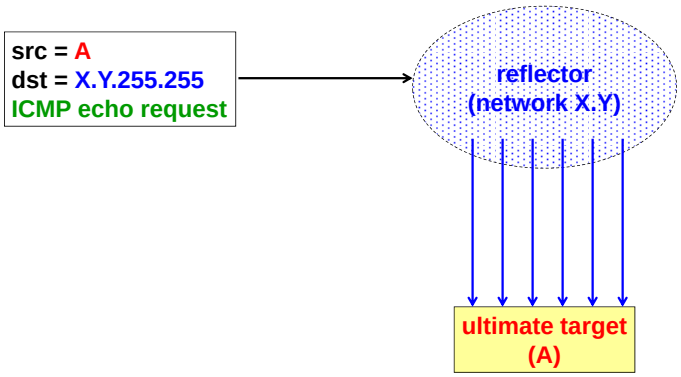
- via a packet source quence (used to slow the transmission of packets when the buffer of an intermediate node are saturated) an attacker can go to decrease the transmission rate of the sender, and then generate a DoS.

- Possibility to use the packages redirect (used as an intermediate node to communicate with another node, the best way forward to deliver the packages) so as to generate an attack Man- In-The-Middle logical.

- Use of a packet time exceeded for a datagram (used when generating a loop in the network) in order to generate a denial of service and bring down the connection between two nodes.

Smurfing attack

Using ICMP is always possible to make a version of ping flooding particularly devastating, which is called Smurfing attack.



In Smurfing the attacker sends an ICMP echo request to a single node, but not to an entire sub-network; the sender of this package will also be the victim (in the example node A). All nodes in the subnet that will receive this message will respond with an ICMP echo reserved respond, flooding the victim of packages.

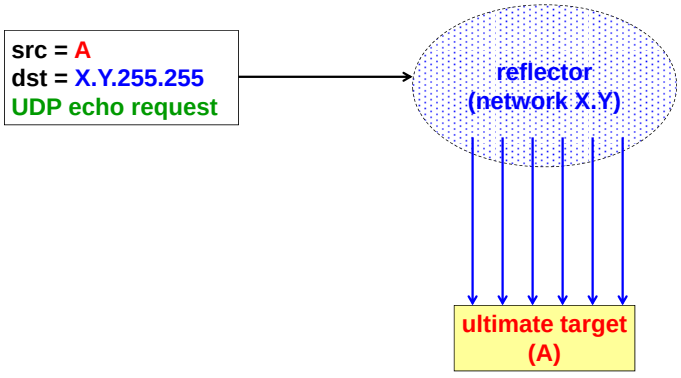
The subnet object package attacker is called reflector, in the sense that bounces messages received. The effectiveness of this attack is that the attacker will only post a package, the receipt of which each node will generate a response that will have as a recipient does the attacker but the victim.

This attack not only because of the problems to the victim, but also because of the problems to the network operator as the huge amount of ICMP packets could go to saturate the bandwidth or otherwise generate big performance issues.

In the past this type of attack is very devastating. With time, however, it was realized that receive broadcast packets from outside the network does not make much sense, and therefore in general IP broadcast packets are rejected if they are generated from internal nodes to the network. If the attacker acts directly from within the subnet the only solution is to use the tools of network management.

5.10.2 Fraggle attack

Following the Smurfing attack has passed to Fraggle attack. This does not exploit the ICMP protocol but the UDP protocol, and in particular one of its small services that allows you to perform the echo request / echo replay. This service mimics the functionality provided by ICMP, but in fact it is used to test the performance and functionality to level 4.



Operation Fraggle attack is similar to the previous one, but this time the attacker sends packets in broadcast type UDP echo request.

This attack is less successful because while ICMP is enabled by default on all network nodes, the UDP small services are activated at the discretion of the system administrator.

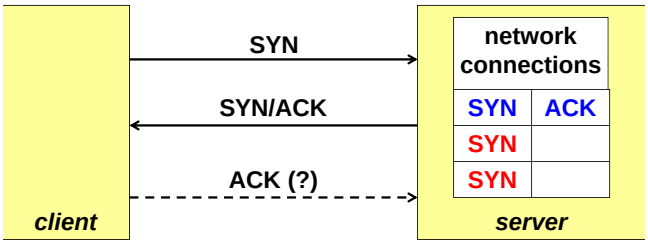
5.10.3 ARP poisoning

This attack has as its object the ARP protocol, and also takes advantage of the fact that this protocol is not authenticated.

In particular ARP poisoning is used to insert false data within the ARP tables, so as to make possible a series of attacks, such as attacks Man-In-The-Middle realized by Ettercap.

5.10.4 TCP SYN flooding

The TCP SYN attack flloding is a very common and very annoying because DoS, which aims to block the application servers.



To create a TCP channel must follow the procedure three-way handshake:

- A client sends a SYN packet.
- Upon receipt of this packet the server allocates a row in the connection table and responds with a SYN / ACK packet.
- The client then responds with an ACK, so that the server can go to complete the line previously created, and then go to open the connection.

In a TCP SYN flooding attack the attacker sends the SYN, but never send the ACK. In particular, the attacker will continue to send SYN packets, which will never follow any ACK.

The effect is that the server will have many rows in the table of open connections only partially filled. This will cause a saturation of the table until the requests for connection will time out (typical 75s). If in the meantime, the server receives a connection attempt from a client, this would not be able to connect, and then you would have in effect a DoS.

Also saw that the attacker does not actually want to connect to the server typically SYN packets are sent with IP spoofing, ie packages will have a false IP address. As a result it will not be easy to find out the identity of the attacker.

Defense from SYN flooding

There are some solutions to try to defend themselves from attack SYN flooding:

- Lower the timeout, but this could result in the risk of deleting valid clients but are slow to create the connection .
- Increase the size of the table, only possible if the server is actually able to manage a large number of connections. Moreover, this solution is avoidable by sending more requests.

- Use a router as SYN interceptor, which is used to replace the server in the first phase. If the handshake is completed successfully then the router will transfer the channel to the server. It has, however, in this case the risk of saturating the router table, solvable problem using timeouts aggressive (with always the risk of deleting valid clients).
- Use a router as SYN monitor, with the aim of killing the connections pending. Even in this case, if the timeout is too low there is the risk of deleting valid clients.

SYN cookie

This idea was conceived by DJBernstein, professor at the University of Chicago. It is the only truly effective way to completely avoid the SYN flooding.

The professor noted that the main problem of the SYN flooding is that servers keep in memory the connection status with various clients, and in the presence of too many requests for connections you run the risk of saturation.

The idea is then to use of SYN cookies, ie use the sequence number of the SYN-ACK packet to transmit a cookie to the client and thus recognize clients who have already sent the SYN without storing anything on the server. This sequence number will be calculated in a cryptographic, or a keyed-digest calculated on the date, time and IP address of the client that sent the SYN.

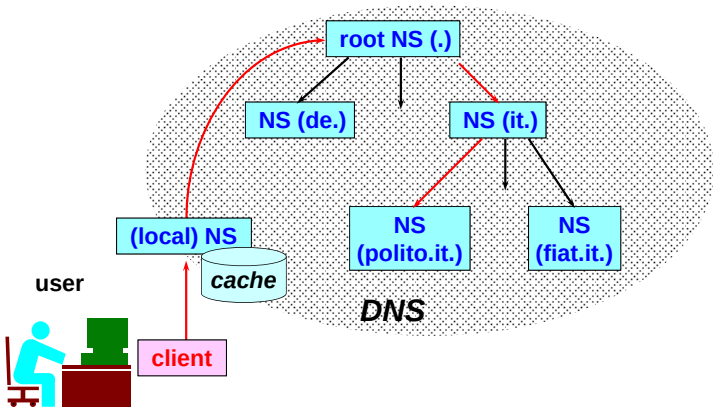
5.10.5 Security DNS

The DNS protocol is the protocol that performs a translation from names to IP addresses and vice versa. The function performed by the DNS is an essential service.

It is subject to several types of attacks, in part due to the fact that for queries uses UDP in part to the use of TCP to transfer to the zones, ie the transfer of large amounts of data from a primary server to a secondary one.

DNS was developed many years ago, and for this reason is devoid of any form of security. In recent years it has tried to develop a secure version of DNS called DNS-SEC.

Architecture DNS



Normally, the DNS is used by a user not directly but through a few program, namely a DNS client that is able to "talk" to the DNS protocol.

The client must be configured to know what are the name-server (NS) of first contact, the so-called local NS. It is defined local because in the case in which the user was in a local network it will also be the NS knows that all nodes in the local network.

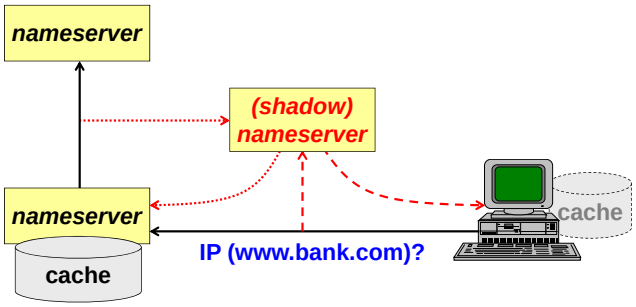
In the case where the user wants to have access to a node outside the local network, the local NS will contact the so-called root NS; NS these are the ones who run the domain'.'. The root NS that receives the user's request will contact the NS first level, who will then query the NS second level and so on. The process continues until the user's request will not be properly translated.

Once the user receives the response to the query, it will be inserted in a cache, so as to avoid a future query. These caches tend to have long life, even ten days.

Shadow DNS server

is one of the attacks easier to do.

Since the packets are not authenticated, if someone were able to intercept a DNS query would be able to provide an answer wrong.



Since queries are based on the UDP client will also get the true answer from NS, but this response is treated as a duplicate packet and then will be discarded. This means that it will be considered the first answer that comes to the client.

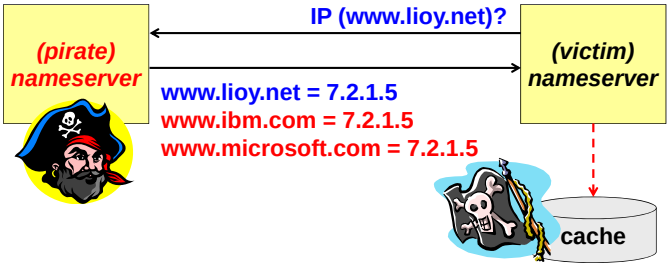
It should be noted that if the shadow NS responds to a query of the client, any other questions of the same type should receive the same answer. If this does not happen, and then the client receives the response from the real-NS, the client should be to obtain the correct translation.

For this reason it is much better to sniff is not on the client-local NS but between geographical, for example, between the local and the root NS NS. so if you go to the local NS provide a wrong answer, this will be cached and therefore will be provided to all clients on the local network that will apply.

It is good to note that the above arguments are based on the fact that only the NS using a cache. However, Microsoft has included on its Windows systems a cache on the client too; this means that the attacks made directly on the local network are most effective against Windows client because it is sufficient to provide the wrong answer once. On non-Windows clients, instead we would need on the local network to continually provide the wrong answer.

DNS spoofing

The previous attack can only take place following a query from the client. For this reason, the attacks have been developed in which the ranging responses to queries that have not been made.



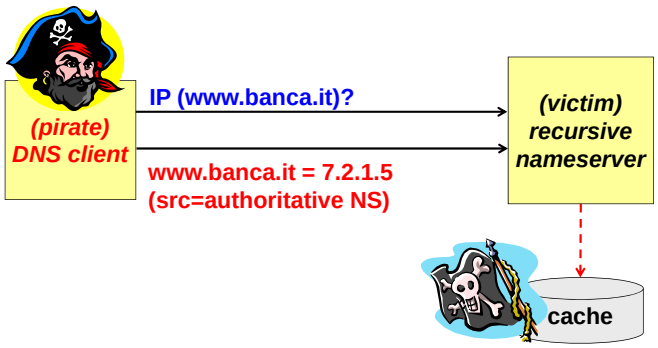
These attacks, which are defined as DNS spoofing, beginning with the lure the victim to make a query on the attacker's NS ("the first to connect to this server will receive as a gift a smartphone"). At this point the client will contact the local NS, asking him to perform the translation domain pirate. The local NS will then contact the NS pirate who will provide the correct answer that will allow the user to connect to the domain pirate. However, though the

response will not only be present this translation, but we will add more name-address incorrect which will also be added to the cache.

In this way you will get answers to queries that the user has never carried out. It is not easy to go to find this type of attack, and for this reason has been particularly effective in the past (some variants still continue to be successful, because it is difficult to go to understand whether the response received is relevant to the question made).

DNS spoofing (v2)

In this second variant the attacker does not even expect that the client make the query is directly the attacker to make the application and to automatically provide the answer.



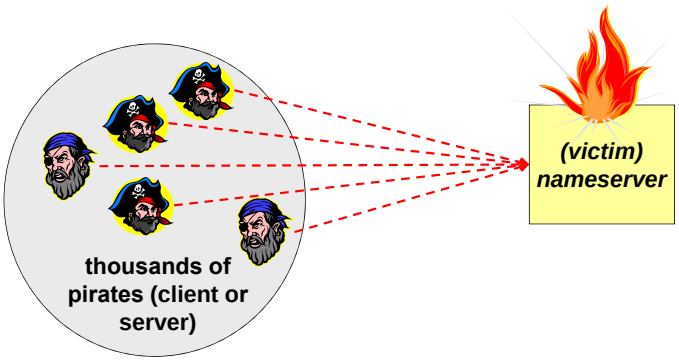
This time then the pirate is no longer a NS but is a client; in particular:

- It will make an application to the local NS victim.
- The local NS but is not aware of the translation, and then will contact NS a higher level. It is at this point that the attacker will send the answer to the question previously made by him, so that the local NS Consider this correct translation. The trick is that the address of this response will not be that of the attacker (local NS would never accept an answer from a client), but rather will be an address obtained through IP spoofing. More precisely, the attacker will pretend to be the authorative NS domain covered by the application.

The response received will be placed in the cache of NS. This implies that all clients will turn to this NS will receive a wrong answer as a result of this name-address translation wrong.

(DNS) flash crowd

A DoS attack runs on DNS (but generally runs on many types of services) is the so-called flash crowd.

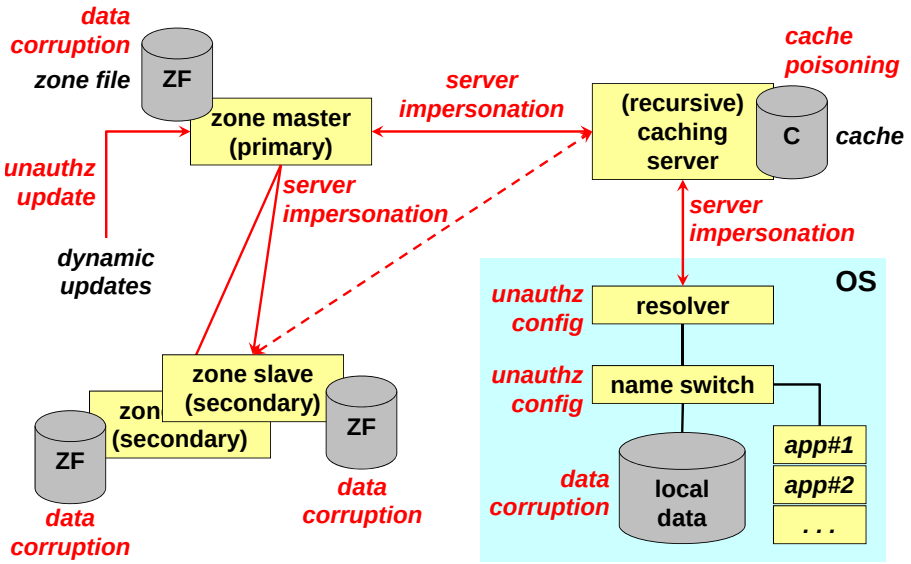


An attack of this kind is to reach agreement among many users of the network and to perform all the same task simultaneously. While the application servers are accustomed to having a large

amount of users, the DNS servers are not designed to receive a large number of requests in a very short period of time.

This attack can then go and make unavailable services do not work not because the servers on which they reside, but because no user is able to know what are the addresses of these servers.

Translation names-addresses



Who asks to resolve a name typically is an application that resides on an OS. Normally inside the SO there is a small process called *name* switch, which allows to resolve names-addresses or addressing data stored locally (data inside the hosts file) or external data (in the case whose name can not be resolved using local data).

If you need to refer to external data that is not direct but through another process called resolver, which has precisely the task of solving the correspondence name-address. The resolver is used mainly because a system may not only use the DNS protocol; is in fact possible to go to use local networks in other systems, such as NIS for Linux systems. These systems provide a translation service but only for the local network.

Suppose, however, that the resolver communicates only with the DNS. The first operation performed is to contact the *local NS*, which is a caching server as it keeps track of all the translations already carried out so as not having to run them again. The local NS is also called *recursive* because once you have contacted the root NS and this has answered, he will always be to contact the NS first-level and lower-level up to what will fail to resolve the name correctly .

The cache server contacts the various levels of NS. at each level you can go talk to a *master (primary)* or with a *slave (secondary)*: each domain of any level must have at least two NS (preferably three, one off-site) . The zone master is the authoritative NS containing the *file* areas, files that contain all the names and addresses of the given area. A copy of this file is also sent, via the transfer with TCP channel, a secondary server, the so-called slave zones. This means that any changes to the names or addresses must be carried out on areas of the file on the primary NS, as the slaves only have copies. In any case when you do a translation request you receive a list of NS to which you can turn; caching server will start at this point to contact all NS until you manage to get an answer.

While in the past every time I needed to add or change something in NS was the systems engineer who went to manually change the zone file, in Windows have been added to the so-called *dynamic updates*. The dynamic DNS updates allow clients to register and dynamically update their associations name-address present on a NS, in the sense that when a client begins to be used it communicates to NS your address so that its association name- address can be updated

accordingly. This is especially useful when a client can move freely within a local network, and therefore can change their address.

This system, however, disrupts the structure of the DNS, as it is no longer true that the tables are changed infrequently, but will be modified each time a client change its address. In addition you will also have permissions problems, as you have to be sure that a client is actually allowed to go to change DNS information.

Given the structure of this system, a series of attacks are possible:

- The user terminal is attacked in the sense that if an attacker manages to have physical access (or even logical for example through a backdoor) could going to change the data stored locally (*date* corruption) or go to edit the configuration data of the name switch or resolver (unathorized configuration). In particular, changing the configuration of the resolver is simple enough to be performed using DHCP, because the protocol not only provides a client IP address but also provides the NS to use for that connection.

- In connection between the resolver and caching server may be one shadow server (*server* impersonation).
  The caching server instead could be run a cache poisoning attack.
  Finally even the caching servers can be personified, in the sense that one can receive answers from a shadow NS. A shadow server in this position has much more effectively than have it in local connection since it causes a change in the cache for all the local domain, while for the local change relates only a single node.

- As for the zone master, if someone could have access to the physical (or logical) to the zone file and eseguirvi changes that would cause a lot of damage, especially since these changes would also propagated to the slave zones. The zone file must then be protected for integrity, to avoid that someone can manipulate it.
  In addition the dynamic updates of Microsoft added a problem of authentication and authorization.

- Slaves rather have the problem dell'impersonation master, because if an attacker impersonates as master and executes a transfer to the slaves would be to change their copies of the zone file. Hence a need for authentication in the transfer areas.
  Finally, since even on the slaves is a local copy of the zone file, it is possible that someone will be able to access them go to modify them at will.

DNSsec: A necessity

Following numerous safety problems related to the use of DNS, we have tried over the years to make sure that protocol. In particular, we have had great changing in February 2008 when a researcher (Dan Kaminsky) has discovered a new attack that makes the cache poisoning much easier to do, much more difficult to avoid and even apply also to records of NS first level.

Only in July 2008 out of the first advisory and the first patch to the problem. At this point, now that the problem had become known, and that the first solutions were starting to come out, Kaminsky in August 2008 is authorized to make public the problem by holding a conference in the Black Hat'08.

Following this, in September 2008 the US make the use of DNSsec for the .gov domain since January 2009.

DNSsec is a system in which all the records that are inserted into the zone files are signed with a digital signature. The digital signature provides integrity (no one can go to edit records at will) but also authentication. However, this introduces two problems:

- What certificates are to be used to make these digital signatures? What PKI can be used?

- The person who has signed a record is authoritative for a domain?

The introduction of the digital signature for each record because management considerably more complex DNS infrastructure because it will have signatures hierarchical, that is when you receive a response must check a whole chain of signatures, and also need to check the validity of proxies, in the sense that you have to be sure that the NS who provided the answer has actually been delegated by the NS higher level. It has also the problem of signatures distributed when you go to use a master and several slave NS: who should be signed, the response?

Delicate is also the management of non-existent names, namely how to behave in case of a translation is requested of a non-existent record. What is done is going to sign also the absence of a record, in order to avoid possible DoS attacks (an attacker could go answer before NS saying that domain is not accessible); This, however, requires that the records are ordered, and that the answers are signed generated on the fly. (Why must be ordered? In DNS records are sorted and www.a.com www.b.com; a user requests the translation of www.azzardo.com. Being ordered the DNS records can prove that there is no domain tried going to provide the ordered list of existing domains. For this reason, the signatures must be made on the fly, because the data to be signed are dynamic.

Although DNSsec is much safer than the simple DNS, its use is still not as frequent as it has some problems:

- You do not have any signs of DNS queries, and thus can be generated false questions.
- There is no root CA (in the sense that there is no single root CA, there are several), but the keys of the root NS are distributed OOB. That list of keys can then be subject to attacks.
- Do not you have no security in the dialogue between the DNS client and DNS server local. It is assumed that this is safe estate here has been in some other way, such as through IPsec, TSIG or SiG (O) (the latter two are used to make authentication and integrity of questions and answers between the DNS client and DNS server local).
- Need to perform encryption directly on the DNS servers. But this generates a computational overhead, but also a management overhead (DNS will become an on-line secure crypto host).
- The record will have a larger size.
- Given the still low use of this solution has not yet had a big trial, and then you have a lot of difficulty finding the correct configurations to ensure a certain level of performance.

5.10.6 Security Routing

The security of the routing is very problematic, primarily because the routing is distributed infrastructure (routers are scattered throughout the territory), which must be managed remotely. Unfortunately, the protocols used for remote management of the router are not particularly safe; such as SNMP, used for network management, is highly uncertain, and easily allows you to manipulate and take possession of network nodes.

Even if you can not directly attack the equipment, you should note that the routers exchange routing tables. Again you have a low level of security for trade because typically authentication is based on IP addresses (just make an IP spoofing). There is the possibility to protect the exchange of these tables by a standard that provides for protection with keyed-digest, and in this way the tables exchanged will be intact and authentic. This solution thus special requires the use of a secret key but shared, and then you have all the problems related to the key-management.

Finally, changes in routing can also be made on the end-node through the improper use of the command ICMP redirect.

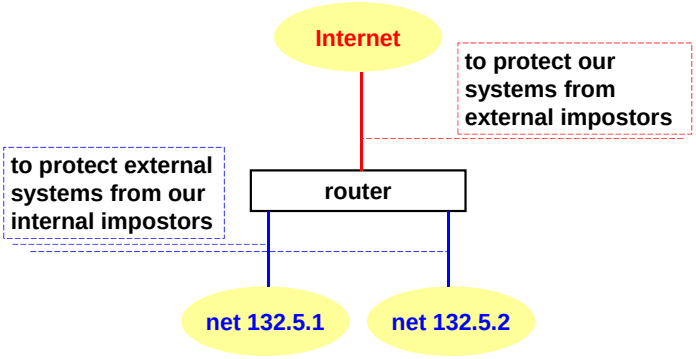
5.10.7 Protection against IP spoofing

For many attacks seen previously is necessary to do IP spoofing. IP spoofing can not be eliminated completely, because anyone can put a false address to a package. But you can try to limit the

problem; to do this, however, manage a network should enable a set of minimum rules to try to protect the users of that network from external impostors, but also to protect the world from impostors internal network. The first is a safety measure almost obligatory, while the second is a measure of the so-called net-etiquette, ie to know how to behave properly in the network in respect of the other nodes.

These safety specifications are contained in RFC-2827, which shows that try to limit the DoS-based IP spoofing, and other auxiliary RFC as RFC-3704 and RFC-3013, which explain how to perform the filtering for multi-homed networks and in general, what are the recommended security measures for ISPs.

Filters for protection against IP spoofing



Suppose you manage the two networks blue. These two networks are connected to a border router, which provides a link to the rest of the Internet.

The traffic to the internet you want to try to protect the internal systems to external networks from impostors, that you want to prevent someone arrivals from providing Internet address as one of the addresses within a network The blue; it is not possible that a package from internet has an internal address as the address to one of the two networks blue. This is the type of filtering is to be done on incoming packets from the Internet.

As for the measures of net-etiquette are to be implemented on the internal interfaces, because these interfaces are to be accepted only packets whose source address an address contained in one of the two networks blue.

5.10.8 Security of SNMP

Many of the problems associated with routing are related to the use of the SNMP protocol. There are several versions of this protocol, in particular the v1 was completely before any level of security, the v2 has a number of security services virtually useless and would v3 security services but are almost never implemented because they are too complex to normal network equipment.

The only thing that is usually made is that the packets are authenticated by the insertion into the package of a shared secret, transmitted in the clear; This secret is called the string "community". Apart from that there is no authentication of the client or the server and no message security.

Chapter 6

Firewall and IDS/IPS

6.1 Firewall

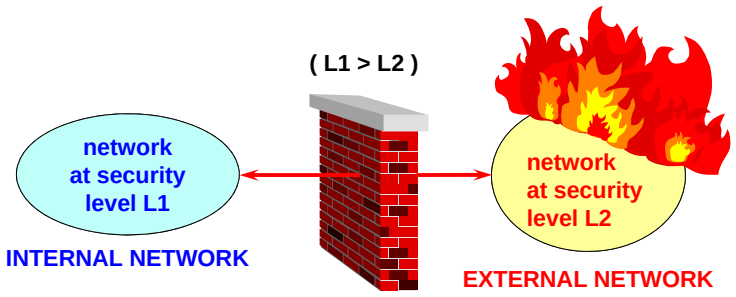


Figure 6.1: The firewall protects the internal network, at a security level L1 greater than security level L2 of the external network.

firewall an internetwork gateway that restricts data communication traffic to and from one of the connected networks (the one said to be "inside" the firewall) and thus protects that network's system resources against threats from the other network (the one that is said to be "outside" the firewall)

The **firewall**, intended as a 'wall against fire' (that is the wall limiting fire propagation from a house to the neighboring house), is a network filter put between two networks having different security levels, which is used to guarantee **boundary protection**: it filters packets in transit, letting just 'good' traffic pass, with the purpose of stopping propagation of attacks coming from the network supposed to be at a lower security level.

Not necessarily the internal network is the corporate network and the external network is the Internet: the firewall can be put even between two network portions within the same company, because attackers could be even inside and the most sensitive areas should be protected.

Firewalls are 100% effective only for attacks over/against blocked channels. The channels through the ports left open require other defenses:

- VPNs (sect. 5.8);
- 'semantic' firewalls (e.g. strong application proxy: sect. 6.3.4);
- IDSes (sect. 6.6) and IPSes (sect. 6.7);
- application-level security (next chapters).

6.1.1 Modes

Firewalls can be configured in ingress and/or egress mode:

- **ingress**: it controls incoming connections, and is used to select which services should be made available to the outside (e.g. web server);
- **egress**: it controls incoming connections, and is used to:
  - prevent internal nodes from connecting to potentially dangerous nodes;
  - check that employees' activity is in line with internal policies (e.g. Facebook filter, leak of corporate secrets).

Distinction between ingress and egress is:

- easy for channel-oriented services (e.g. TCP): the connection has been set up by one of the two parties (e.g. three-way handshake);
- difficult for datagram-based services (e.g. UDP, ICMP): the concept of connection does not exist.

For example, implementing the ingress firewall is troublesome in the case of the FTP protocol: although who opened the FTP connection is the user from the internal network, the channel for file transfer is always opened by the external server.

6.1.2 Basic elements

Basic elements which make a firewall are:

- **screening router** (choke) (sect. 6.5.1): it is a router which, besides routing, filters traffic at IP layer;
- **bastion host** (sect. 6.2.4): it is a particularly secure and protected system, with auditing functions (e.g. logging);
- **application-level gateway** (proxy) (sect. 6.3.4): it is a service which works on behalf of a certain application, by filtering traffic at application layer (e.g. HTTP) and typically also performing access control;
- **dual-homed gateway** (sect. 6.5.2): it is a system having two network cards, interfaced on two different networks, to play the role of bridge between a network and another, but with routing disabled.

6.2 Firewall design

Firewalls are designed, are not bought: a firewall is not a single object, but is made up of several components which can be bought. Design has the purpose of finding an optimal tradeoff between security and functionality, by selecting the proper components and keeping costs as low as possible.

How to filter by stopping just 'bad' traffic and allowing just 'good' traffic? The ideal firewall would be the one completely cutting the link between the internal network and the external network, but this would make the network useless $\Rightarrow$ a tradeoff needs to be found between functionality and security: the more packets are allowed to pass, the greater the probability of attack propagation will be.

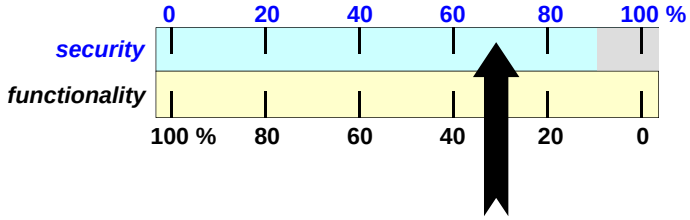


Figure 6.2: The security index.

6.2.1 The three firewall commandments

1. The firewall must be the only contact point of the internal network with the external one.
   The boundary around the internal network must not have 'breaches' toward the external world (e.g. an employee is connected at the same time to the corporate network and to the mobile network by an Internet key).

2. Only the 'authorized' traffic can cross the firewall.
   Who designs a firewall must have a list of services which have to be made available.

3. The firewall must be a highly secure system itself.
   Other software which could be subject to bugs and vulnerabilities must not be installed on the machine where the firewall is running.

6.2.2 Authorization policies

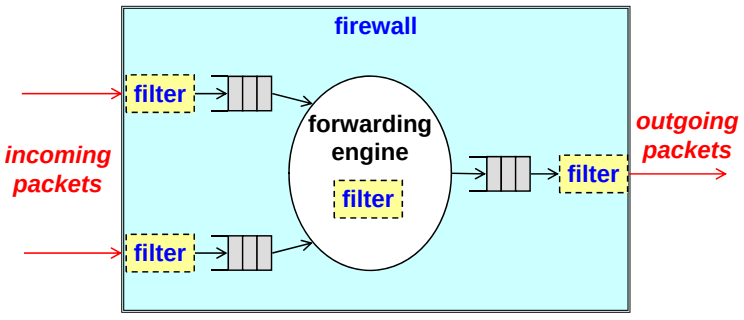
Whitelisting 'Everything is not explicitly allowed, is forbidden'
A whitelist lists the authorized services, and all the services not included in the whitelist will be blocked:

- greater security: an unexpected kind of attack will fail;
- more difficult to manage: users' actual needs need to be understood in order to avoid blocking essential services for the company.

Blacklisting 'Everything is not explicitly forbidden, is allowed'
A blacklist lists the unauthorized services, and all services not included in the blacklist will be authorized:

- lower security: all possible attacks are difficult to foresee;
- easier to manage: users are more free.

6.2.3 Filtering points



A forwarding engine processes the packets queued in input and forwards them into output queues. Choosing the filtering position affects firewall performance:

- incoming packets: usually it is the best position:
  - + immediate filtering: the forwarding engine has to process less input data;
  - − multiple input queues: multiple filtering processes in parallel are needed;
- forwarding engine: it is not so bad:
  - + more information: it is good if the filter is based more on routing rules rather than source and destination addresses;
- outgoing packets: generally it is the worst position:
  - − computational resources: all packets need to be processed by the forwarding engine.

6.2.4 Bastion host

> **bastion host** a strongly protected computer that is in a network protected by a firewall (or is part of a firewall) and is the only host (or one of only a few) in the network that can be directly accessed from networks on the other side of the firewall

The bastion host should be a particularly secure and protected system:

- dedicated machine: the bastion host should not be used by users or to install other software, but should be used exclusively for its function of firewall;
- software:
  - minimal configuration: it is better to just leave essential services to guarantee the proper firewall operation, without running useless processes which could have bugs which could be exploited to perform attacks;
  - keep it simple (KISS): the number of bugs grows exponentially with the number of line codes ⇒ it is better to split into a lot of smaller components, easier to design, implement and monitor;
  - only certified components: unknown software could include malware;
- auditing: it should save the log of all activities, better if on a remote, secure server inside the network and exclusively used for this purpose;
- no source routing: the source should not affect the packet paths, otherwise they could bypass the block;
- no IP forwarding: only packets properly checked and filtered should go from an interface to the other one;
- traps for intruders: for example the `ls` function is changed so that it sends an alert to the system administrator.

6.2.5 DMZ

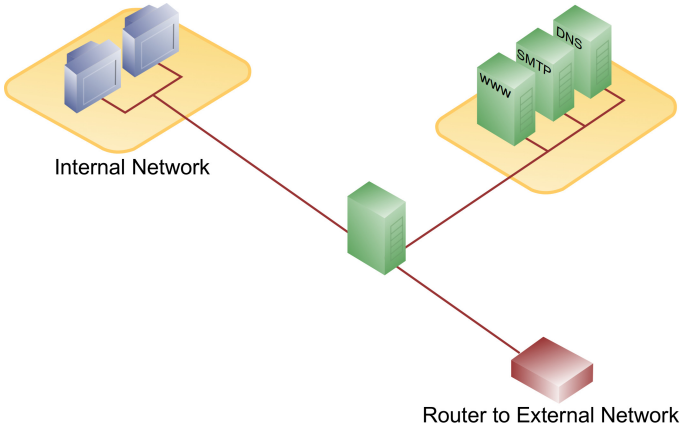


Figure 6.3: Three-legged firewall.[1]

buffer zone a neutral internetwork segment used to connect other segments that each operate under a different security policy

In a corporate network there could be some servers which offer public services to users in the external network (e.g. server hosting the corporate web site). If these public servers were put inside the internal network, an attacker from the external network could exploit a vulnerability on the server (e.g. bug in PHP modules in the web page) to gain direct access to the internal network.

The solution is to put all servers which should be reachable from outside not into the internal network, but into a network segment isolated from both the internal network and the external one, called **De-Militared Zone** (DMZ):

- a possible attack does not propagate to the internal network where there are hosts;
- public servers are still protected by the firewall;
- routing can be configured so that the internal network is completely unknown to the outside network.

On the market firewalls having even more than three legs are available: the additional legs are used to have multiple DMZs, so as to further segment objects and prevent servers performing different functions from directly communicating (e.g. a DMZ for the financial server, another one for the design server).

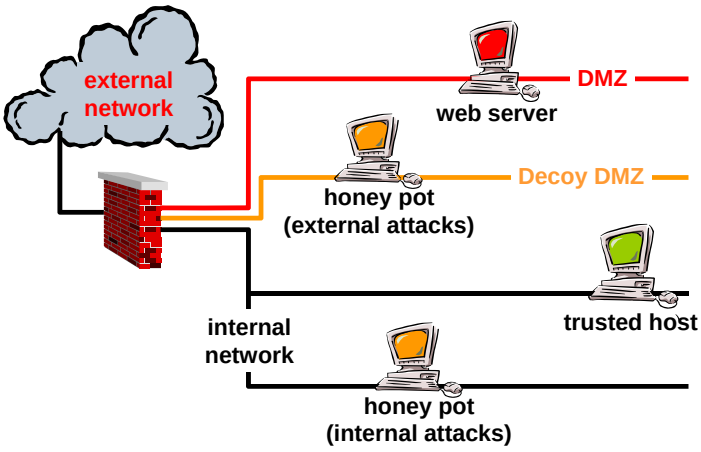
Honey pot

honey pot a system (e.g., a web server) or system resource (e.g., a file on a server) that is designed to be attractive to potential crackers and intruders, like honey is attractive to bears

A lot of companies, besides having a normal DMZ, have a second DMZ, often called **decoy DMZ**, on which fake servers being intentionally easy attack targets (e.g. username = 'root', password = 'root') are put. The purpose is to attract attackers so as to analyze their behaviours and even manage to discover their identities.

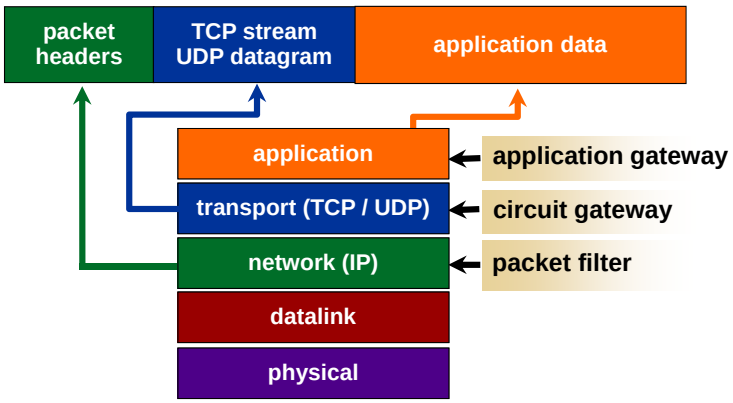
The firewall may have holes toward the 'honey'. A honey pot can be also put in the internal network against attacks coming from inside.

[1] This image is taken from Wikimedia Commons (DMZ network diagram 1.svg), it was made by user Sangre viento, by Jason Funk and by user Pbroks13 and it is in the public domain.



Honey net The honey net is a even wider concept: a whole fake corporate network is created with the only purpose of collecting information. Very often these networks are created by antivirus developers: attackers, by infecting these controlled networks, provide information about new attack ways and new malware.

6.3 Firewall classification



At which layer are checks made?

- the lower the layer is, the greater the filtering speed is: processing is simpler (at layer 3 one packet at a time is processed);
- the higher the layer is, the greater security is: more information is available on which filtering decisions can be taken.

From the commercial point of view several flavors exist which can be classified into three types of firewalls:

- **packet filter** (sect. 6.3.1): filtering can only be by header in each packet (network layer);
- **circuit-level gateway** (sect. 6.3.3): filtering is by TCP/UDP flow (transport layer);
- **application-level gateway** (sect. 6.3.4): filtering is by payload (application layer).

From the technical point of view, the main differences among these systems are in terms of:

- performance;

- protection of the operating system on the firewall itself;
- compliance with client-server model.

When two security devices (e.g. a packet filter and a gateway) are put in a series, it is better that they are from different vendors: if the software in the first device had a bug, even the second one would be very likely to have it if both of them are based on common libraries ⇒ an attacker can just exploit this bug to bypass both devices.

6.3.1 Packet filter

The **packet filter**, historically available on routers, performs inspections on single IP packets based on:

- IP header: addresses;
- TCP/UDP header: ports.

Advantages

- excellent scalability: a packet filter just parses headers independently of application protocols;
- excellent performance: it can be implemented in hardware;
- low cost: it is a functionality available on all routers and a lot of operating systems.

Disadvantages

- stateless: each packet is parsed independently of the other ones;
- IP fragmentation: the packet could lack the TCP/UDP header;
- security: checks are little precise and so easier to fool (e.g. IP spoofing);
- complex configuration: it is hard to deal with numbers rather than names;
- there are issues in supporting services based on port dynamic allocation (e.g. FTP).

6.3.2 Stateful (dynamic) packet filter

The **stateful** (dynamic) **packet filter** is "state-aware":

- it stores state information from the transport layer and/or the application layer;
- it distinguishes the new connections from the already open ones, thanks to a state table for open connections ⇒ packets matching one row in the table are accepted without any further check.

Example The packet filter temporarily opens the FTP port only when a file transfer starts by the `PORT` command.

Advantages

- stateful: decisions are better because based on state information;
- parallelization: multiple CPU cores work in parallel (Symmetrical Multi-Processing [SMP]).

Disadvantage A lot of intrinsic limitations of the packet filter remain.

6.3.3 Circuit-level gateway

The **circuit-level gateway** is a non-'application-aware' proxy:

- it creates a transport-layer circuit between the client and the server;
- it have no knowledge of transit data syntax.

Advantage Servers are isolated from all attacks concerning:

- TCP three-way handshake: protection switches on at the beginning of the session and lasts for the whole session;
- IP packet fragmentation: the proxy re-assembles the packet to completely know its content.

Disadvantage

- client-server model break (within the session duration): it may require changes to applications (e.g. client authentication: normally it occurs at application layer, not at transport layer);
- a lot of intrinsic limitations of the packet filter remain.

6.3.4 Application-level gateway

The **application-level gateway**:

- inspects packets at application layer (payloads): it is made up internally of a set of proxies, one for every application protocol;
- can play the role of terminator: it directly interacts with the client as if it was the server, and with the server as if it was the client;
- can perform IP address masking or re-numbering: it can be needed if it is acting as a terminator and one of the two parties asks the other one for authentication;
- can also have authentication features, especially in egress: the gateway asks the client in the external network to authenticate itself so as to apply to it the proper policies.

Advantages

- maximum security: rules are more fine-grained and simpler with respect to the packet filter, because checks are based on the application layer;
- parallelization: multiple CPU cores work in parallel (SMP).

Disadvantages

- performance: deeper checks take more time;
- application-dependent: each application protocol requires a specific proxy:
  - delay in supporting new applications;
  - resource consumption: a lot of proxies mean a lot of processes;
  - low performance: processes work in user mode;
- client-server model break: it may not be altogether transparent to clients, and often requires a change to the client application;
- attacks: since the client directly interacts with the gateway, the operating system in the firewall is exposed to attacks;
- application-layer security protocols: the firewall could be unable to correctly parse the packet content (e.g. SSL: the packet content is encrypted).

Variants

- **transparent gateway**:
  - \+ transparent: it is less intrusive because it does not require configuration on the client;
  - − implementation complexity: the system complexity is moved from the client to the router, which will have to re-route packets toward proxies;
- **strong application proxy**: checks are based on the semantics and the policy, not just the syntax of the application protocol:
  - syntax: is the `GET` command in the proper protocol format?
  - semantics: is the `GET` command asking for a really existing resource?
  - policy: is the `GET` command asking for a resource authorized by the policy?

6.3.5 Reverse proxy

The **reverse proxy** is an ingress filter put immediately before one or more HTTP servers: it only acts as a front-end toward the client, pretending to be the server, and then delivers requests to the actual server.

Features

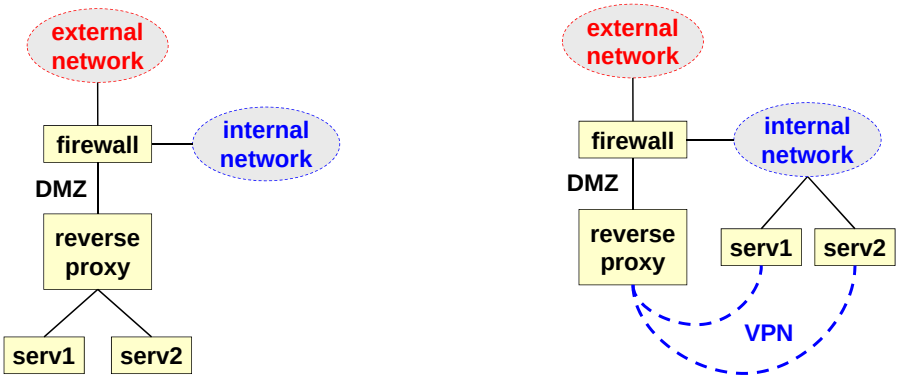
- content inspection: it checks every request before delivering it to the server;
- Access Control List (ACL): it decides which addresses can connect and which protocols can be used;
- obfuscation: the proxy does not tell the type of server actually used, protecting it against some kinds of attack;
- SSL accelerator: it decreases the number of cryptographic operations which the server has to perform improving performance, but communications between the reverse proxy and the server will be unprotected (at least at protocol level);
- load balancer: it selects the least loaded server, guaranteeing a better robustness against DoS attacks;
- web accelerator: it locally caches static contents requested more frequently, so as to decrease the server load;
- compression;
- spoon feeding: it locally copies a whole, dynamically created page, and sends it little by little to the client ⇒ it is useful when the user is connected to a very slow network (e.g. mobile network) to unload the application server.

Architecture

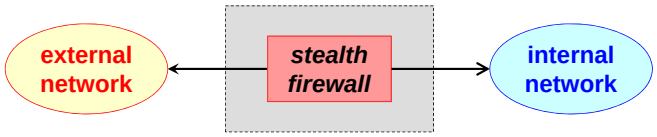
From the architectural point of view, the reverse proxy is compulsorily put on the DMZ channel because it will be reachable from outside.

What if servers needs to access a database in the internal network? Two configurations are possible:

- servers are in the DMZ, physically attached to the proxy, and have return channels toward the database in the internal network ⇒ accesses to the database are very slow;
- servers are in the internal network, hence closer to the database, and the proxy communicates with servers across a VPN (e.g. end-to-end IPsec).



6.3.6 Stealth firewall



The **stealth firewall** is a firewall with no network addresses: it physically intercepts transit packets, by putting its network interfaces in promiscuous mode, and if they match the security policy it will forwards them, totally unchanged, to the other network interface.

Its presence is invisible to the client: it can not be the addressee of any packet, therefore can not be attacked directly. At most a transmission delay increase can be noticed, depending on the time required by checks.

Not all firewalls can be used in stealth mode:

− application-level gateway: it needs to have an address being based on a client-server communication;

− circuit-level gateway: it needs to have an address having to work as a terminator;

\+ packet filter: it just has to look inside packets.

6.3.7 Local firewall and personal firewall

The firewall can be directly installed on the node to be protected:

- **local firewall**: it is a firewall, especially ingress, installed on the server;
- **personal firewall**: it is a firewall, especially egress, installed on the client.

Being directly installed on the node to be protected, it runs inside the operating system and therefore can check not just protocols and ports, but also which processes are allowed to:

- act as a client (egress): open network connections towards other nodes (e.g. a malware could send data to remote nodes and propagate to them);
- act as a server (ingress): receive connection or service requests.

6.4 SOCKS

SOCKS, also known as 'Authenticated Firewall Traversal' (AFT), is the most widespread circuit-level gateway in the world. The big success of SOCKS comes from the fact that it is supported commercially by both browsers (e.g. Firefox, Internet Explorer) and several firewalls (e.g. IBM).

In order to be able to authenticate clients at transport layer, clients need to be changed by the SOCKS open-source library:

- it includes standard clients (e.g. Telnet, FTP, Finger, Whois);
- it provides a library to develop own clients.

Client side The library replaces the standard functions which deal with sockets (`connect()`, `bind()`, `accept()`, ...) with functions which have the purpose of:

- opening a channel with the SOCKS server;
- sending:
  - the protocol version number;
  - the IP address and the port the client wants to connect to;
  - the username of the user requesting the operation.

Server side The SOCKS server:

- checks its ACL to know if the user has the rights required to open a connection toward the specified address and port;
- if yes, opens the requested channel, but with its own IP address ⇒ the SOCKS server receives data from the client, re-assemblates them and forwards them toward the destination server (vice versa for replies).

6.4.1 SOCKS 5

SOCKS 4 had the following issues:

- it could not distinguish between internal and external network;
- user authentication was very weak (based on a daemon called `identd` which accepted what the client was claiming);
- it only supported the TCP protocol.

Versione 5, the first one developed inside IETF, made several improvements:

- UDP support;
- distinction between internal network and external network;
- better authentication (username and password, or system similar to Kerberos);
- encrypted channel between SOCKS client and server.

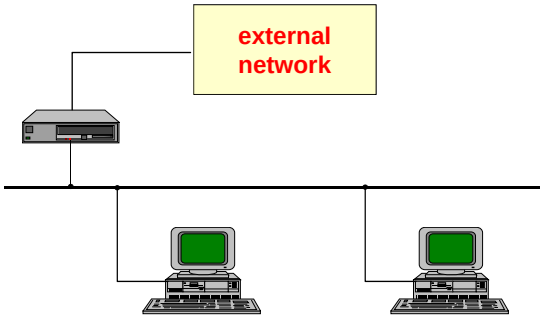
6.5 Firewall architectures

6.5.1 Screening router (choke)

> **screening router** an internetwork router that selectively prevents the passage of data packets according to a security policy

A single device, called **screening router**, is put between the external network and the internal network, performing the functions of:

- router: routing;
- packet filter: traffic filtering at single packet level (addresses, ports, protocols).



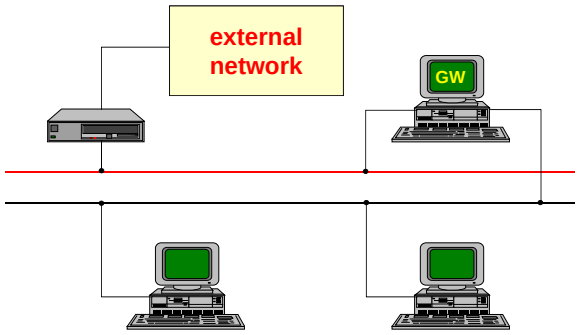
Advantages

- no proxies: no changes to applications are required;
- simplicity: it is the simplest architecture;
- cost: no dedicated hardware is required.

Disadvantages

- security: it only checks at network layer;
- single point of failure: the router firmware could have bugs;
- no DMZ: public servers could be source of attack in the internal network.

6.5.2 Dual-homed gateway



A **dual-homed gateway**, a machine having two network cards, one interfaced to the external network and the other one interfaced on the internal one, both of them with disabled routing, and a running process which is in charge of deciding which traffic is authorized to go from a network interface to the other one, is put in series with the screening router.

This system has a double check point:

- the router performs the function of packet filter;
- the dual-homed gateway hosts a circuit-level gateway or a application-level gateway depending on needs.

Since the gateway is interfaced on two different networks, the intermediate network is isolated from the internal network and can be used as a DMZ for public servers.

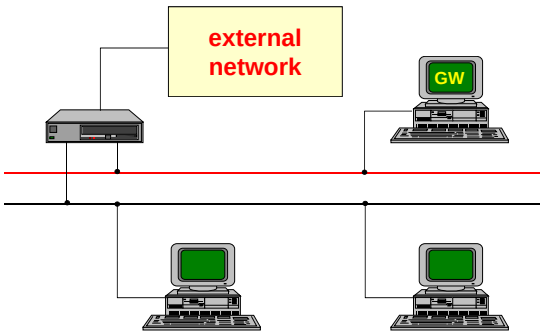
Advantages

- masking: the gateway can mask the internal network;
- simplicity: the implementation is simple;
- cost: just small additional hardware requirements are required (e.g. fast network card);
- double defense line: exploiting the bug of one of the two devices is not enough to enter the internal network.

Disadvantages

- management: two systems are needed;
- bottleneck: all traffic has to cross the dual-homed gateway;
- flexibility: some kinds of traffic require to be checked by servers in the internal network (e.g. spam e-mails).

6.5.3 Screened host gateway



The bridge between the internal network and the external network is moved to the screening router, to which a network card is added: once the packet coming from outside has been authorized by the packet filter, it can follow two paths:

- it still can be sent to the gateway (now become a bastion host), if it needs further investigation;
- it can directly enter the internal network, if it needs to be checked by an internal server (e.g. mail server).

Advantage flexibility: checks related to some services or hosts can be made lighter by avoiding to cross the gateway.

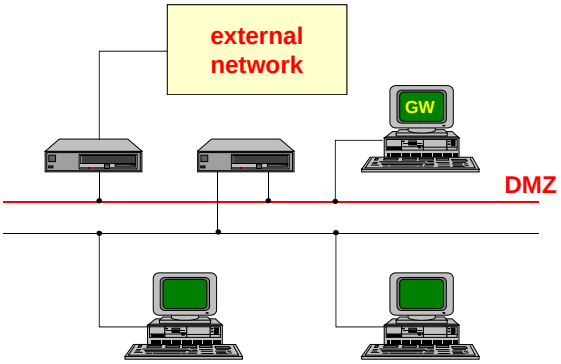
Disadvantages

- masking: masking is possible just for packets crossing the bastion host;
- single point of failure: the router firmware could have bugs.

6.5.4 Screened subnet

The screening router has been split in two:

- the packet filter part filters incoming and outgoing traffic;
- the router part performs the function of bridge between the two networks.



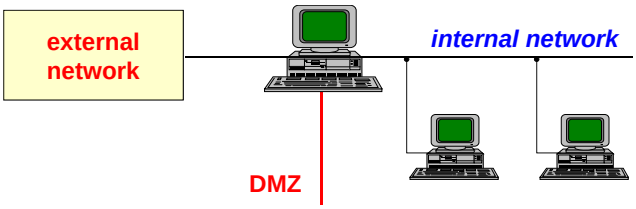
Advantages

- flexibility: checks related to some services or hosts can be made lighter by avoiding to cross the gateway;
- double defense line: exploiting the bug of one of the two devices is not enough to enter the internal network.

Disadvantages

- cost: three devices are needed;
- multi-vendor: devices should be by three different vendors, otherwise they could suffer from common bugs.

6.5.5 Screened subnet with three-legged firewall



In order to simplify the implementation of the screened subnet architecture while keeping its functionality, a lot of companies have proposed to embed the functions of the two routers/packet filters into the gateway. The gateway has three network cards, interfaced each to three different networks (**three-legged firewall**): the external network, the internal network and the DMZ.

Advantages lower cost: three functions are concentrated on a single physical device.

Disadvantages single point of failure: three processes are running on the same operating system.

6.6 IDS

intrusion detection sensing and analyzing system events for the purpose of noticing (i.e., becoming aware of) attempts to access system resources in an unauthorized manner

Intrusion Detection System (IDS) system to identify individuals using a computer or a network without authorization

An **Intrusion Detection System** (IDS) monitors the network to:

- identify unauthorized users;
- identify authorized users, but violating their privileges.

How to distinguish attack packets from normal packets? Detection is based on behavioural 'pattern': an attacker is assumed to behave differently from a normal user.

6.6.1 Classification

Functional features

From the functional point of view, IDSes can be distinguished into:

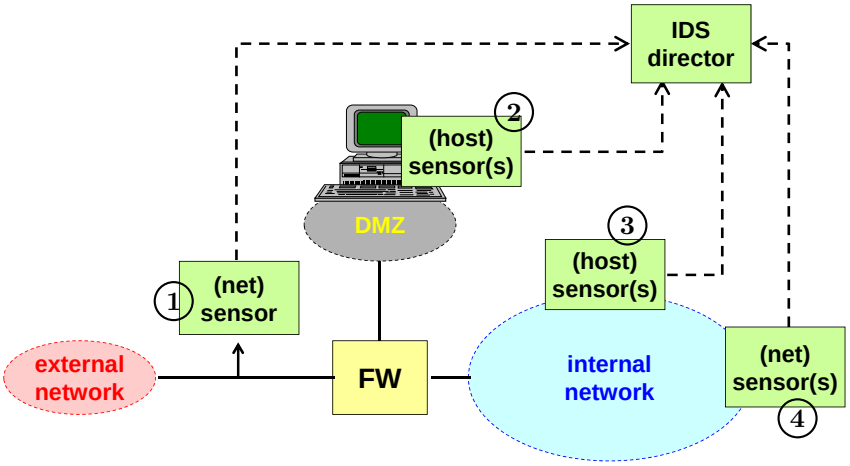
- **passive**: they have a reactive approach: they detect an intrusion after this has happened:
  - cryptographic checksums: has the file been changed?
  - patterns ('attack signature'): does the file contain the virus signature?
- **active**: they have a proactive approach: they detect an intrusion while it is ongoing:
  1. learning: statistical analysis of the system operation in order to learn the normal behaviour of the system (e.g. traffic distribution over week days, day hours, etc.);
  2. monitoring: active analysis of the system operation, and comparison with the behaviour expected by statistics searching for anomalies (e.g. ICMP packet percentage increase);
  3. reaction: a reaction is triggered when a certain threshold has been exceeded (e.g. ICMP packet excess): an attack could be ongoing or not, also depending on where the threshold has been placed.

Topological features

From the topological point of view, IDSes can be distinguished into:

- **host-based** (HIDS): they are based on enabling internal monitoring tools in the operating system:
  - System Integrity Verifier (SIV) (e.g. Tripwire): it monitors the file system to detect file changes (e.g. Windows registries, cron [Linux scheduler] configuration, user privilege change);
  - Log File Monitor (LFM) (e.g. Swatch): analysis of logs of the operating system or even applications, to detect known patterns resulting from attacks or attack attempts (e.g. the user failed his password n times);
- **network-based** (NIDS): they are based on enabling network traffic monitoring tools:
  - sensors: they are spread all around the network to be monitored:
    - they check traffic and log detecting suspected patterns;
    - they generate security events in case of anomalies;
    - they can interact with the system (e.g. sending TCP resets to close channels, changing ACLs);
  - director: it is the core node which aggregates information coming from sensors:
    - it coordinates the activity of single sensors;
    - it decides whether the anomaly is an attack based on data in the security database;
  - IDS message system: it allows a secure and reliable communication among IDS components.

6.6.2 NIDS architecture



Where to put sensors?

1. network sensor before the firewall: it provides information about overall attacks, even the ones which will be stopped by the firewall ⇒ it is useful to know statistical information about attacks which the firewall is able to stop;
2. host sensor(s) in the DMZ: they are used to monitor the public servers hosted on the DMZ;
3. host sensor(s) in the internal network: they are used to monitor critical servers (e.g. financial, design) in the internal network;
4. network sensor(s) in the internal network: they are used to:
   - detect whether somebody from outside was able to bypass the firewall;
   - monitor employees' misbehaviours in the internal network.

6.6.3 IDS/NIDS interoperability

Not necessarily all IDS components come from the same vendor ⇒ common solutions are needed:

- attack signatures: currently no standards exist for signature format, but the Snort format is very spread;
- alerts: two solutions are competing for alert format and transmission protocol:
  - IDMEF + IDXP + IODEF: three standard protocols proposed by IETF;
  - SDEE: protocol pushed by some companies (e.g. Cisco).

6.7 IPS

Intrusion Prevention System (IPS) system that can detect an intrusive activity and can also attempt to stop the activity, ideally before it reaches its targets

An **Intrusion Prevention System** (IPS) is an automatic system which is used to speed up and automate reactions to intrusions:

- IDS: it just alerts an attack is ongoing;
- IPS: it tries to prevent attacks.

An IPS is made up of:

- IDS: it detects an ongoing attack (e.g. ICMP packet excess);
- distributed dynamic firewall: it reacts very quickly (e.g. stopping all ICMP traffic).

Disadvantages

- since decisions are automatic, an IPS could take wrong decisions stopping harmless traffic;
- it is not a product but a technology, with a large impact on a lot of elements in the protection system.

Chapter 7

Security of network applications

Typically developers of network applications do not consider some security aspects:

- weak authentication issues:
  - password snooping: when the authentication is only based on username and password without using protected channels;
  - IP spoofing: when the authentication is based on IP addresses;
- other frequent issues:
  - data spoofing: data can be read;
  - data forging: data can be manipulated;
  - shadow server and man-in-the-middle attacks;
  - replay attacks.

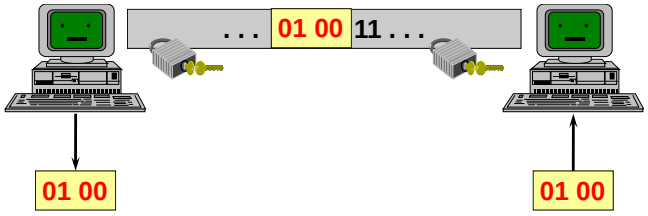
7.1 Security approaches

Two approaches, theoretically opposing but actually complementary, can be adopted in order to try to solve these security issues:

- channel security (sect. 7.1.1): all data is protected only during the transit in the channel;
- message security (sect. 7.1.2): only enveloped data is always protected.

The two approaches can be combined by transmitting protected confidential data over secure channels.

7.1.1 Channel security



Security is made at level of network channel: before the transmission the two peers set up a secure channel (e.g. IPsec, TLS), and all data which will cross that channel will be protected, even if it has not been encapsulated into secure envelopes.

Security properties

+ integrity: if data has been altered, the receiver will not be able to understand it in decryption;
+ authentication (single or mutual): the channel is negotiated between the two peers;
+ confidentiality: the channel is encrypted;
− non repudiation: data can not be digitally signed by the sender.

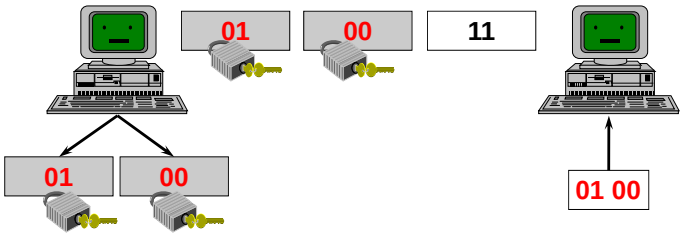
Advantages At most small modifications to applications are required:

- IPsec: it does not require any modification because it is installed at IP level;
- TLS: socket calls need to be changed to TLS calls.

Disadvantages

- enforced protection: all transit data is protected, even the one which does not need protection;
- temporary protection: data is protected only during transit in the channel, and will become insecure again when it exits the channel;
- non repudiation: the two peers have to trust each other.

7.1.2 Message security



Security is made at level of data: confidential data is encapsulated into secure envelopes (e.g. PKCS #7[1]), and can be sent even over insecure channels.

+ integrity: the receiver will be able to detect an alteration by means of various mechanisms (e.g. invalid digital signature, non-matching keyed-digest);
- authentication:
  + single: the recipient receives from the sender envelopes on which a variety of mechanisms (e.g. digital signature, keyed-digest) providing sender authentication can be applied;
  − mutual: the recipient does not send to the sender any data on which to apply mechanisms providing authentication;
+ confidentiality: envelopes are encrypted;
+ non repudiation: envelopes are digitally signed by the sender.

[1] Please see section 3.10.

Advantages

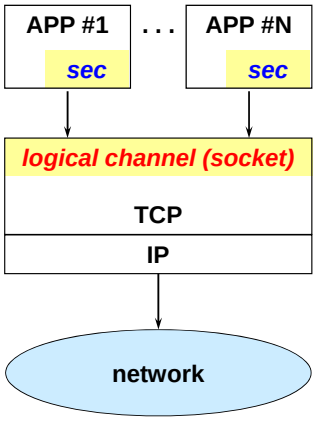
- non-enforced protection: only actually confidential data can be protected, while non-confidential data will be inserted into normal packets;
- permanent protection: data keeps being protected even outside the channel (e.g. stored on disk);
- non repudiation: the sender can not deny of having digitally signed the data.

Disadvantages Modifications to applications are required:

- sender side: the application generating/sending the data has to encapsulate it into an envelope;
- receiver side: the application receiving/processing the data has to extract it from the envelope.

7.2 Security system implementation

7.2.1 Internal security inside applications



Each application has its own security part directly implemented inside itself by the developer. The common part among the various applications is limited to communication channels (sockets).

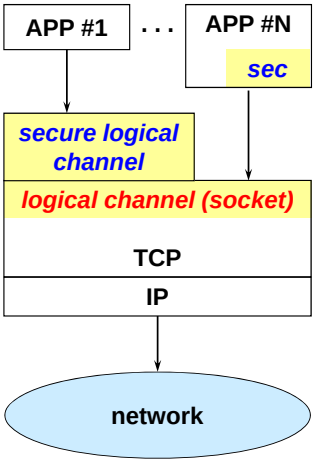
Advantage message security: only the application knows the format of transmitted and received data

Disadvantages

- implementation mistakes: security protocols free of vulnerabilities are not easy to invent;
- interoperability: a protocol is not guaranteed to have been implemented in the same way by different applications.

7.2.2 External security outside applications

Applications also share the security part: between the TCP layer and the application layer there is a common **secure session** layer which makes the network logical channel secure, independently of the application which requested to create it.



The ideal layer in the OSI stack would be the session layer (layer 5), because a session is just a logical channel, but unfortunately this layer in the TCP/IP stack is embedded in the application layer ⇒ the secure session layer is an abstraction layer constituted by code in a shared library extending the socket interface.

A lot of competing protocols exist to create secure channel:

- SSL/TLS (sect. 7.3): it is the most widely used protocol;
- SSH: it was a successful product in the period of restriction to cryptographic material export from USA, but today it is mostly a niche solution;
- PCT: it was proposed by Microsoft as an alternative to SSL, but it has been a total fiasco (even Microsoft abandoned it long ago).

Advantages

- ease: the developer just has to tell he wants to open a secure socket channel, by specifying the desired security parameters;
- implementation mistakes: the developer does not need to develop by himself the security library;
- on application choice: the developer is not forced to use the extended socket interface, but can use a normal socket and implement his own security library.

Disadvantage channel security: the library opens a secure logical channel to transmit and receive bit sequences

7.3 SSL

Secure Socket Layer, first proposed by Netscape, is a protocol to create secure transport channels.

SSL lies approximately at the session layer, between the transport layer and the application layer:

- it can be used with any application-layer protocol (e.g. HTTP, SMTP, NNTP, FTP, TELNET);
- it requires a reliable transport protocol which provides the SSL layer with non-out-of-sequence and non-missing packets (e.g. TCP).

7.3.1 Security features

- **server authentication** (mandatory): when opening the channel, the server:
  1. authenticates itself by sending its own public key, typically included inside an X.509 certificate;
  2. it undergoes a sort of asymmetric challenge to prove he knows the private key corresponding to the public key in the certificate;
- **client authentication** (optional): when opening the channel, the client authenticates itself in the same way as the server;
- **message authentication and integrity** (mandatory): all transmitted data is protected by keyed-digest MAC (e.g. SHA-2) (the digital signature would be too slow for data streams);
- **message confidentiality** (mandatory in SSL 2, optional in SSL 3): all transmitted data is encrypted by a symmetric session key (e.g. 3DES):
  - the client decides the key;
  - the client communicates the key to the server by means of public-key cryptography (e.g. RSA, DH);[2]
- **protection against replay and filtering attacks**: each message is numbered by a MID[3], guaranteeing a full protection possible thanks to the reliable transport protocol (not partial like in IPsec[4]):
  - replay: messages have to always arrive in the same order they have been sent and they have not to be duplicate;
  - filtering (= cancellation): no messages have to be missing.

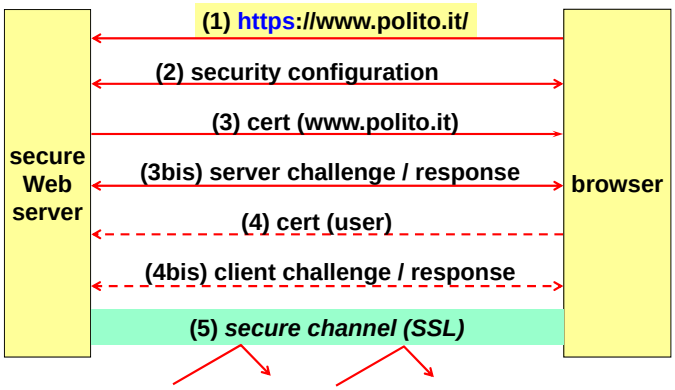
Some security features are mandatory and other ones are optional:

- peer authentication:
  - server authentication: it is mandatory because it is fundamental to defend against shadow server and man-in-the-middle attacks when the user performs purchases on the Internet;
  - client authentication: it is optional because:
    - it can be performed also at application layer (e.g. username and password);
    - the user usually has not a certificate of his;
    - in electronic commerce it is important that the user pays, not his identity;
- message authentication: according to several studies 10% messages actually needs confidentiality (optional in SSL 3), while 100% needs authentication and integrity (mandatory).

7.3.2 Conceptual scheme

1. The user through a browser connects to a secure server.
2. At this point the browser and server must negotiate a security configuration, just because you can go to many algorithms use both the server side is the client side . The result of this phase could also be that the server and the client does not support the same type of protocols, and then you can not establish a secure connection.

[2]Please see section 2.3.1.
[3]Please see section 2.9.3.
[4]Please see section 5.9.9.

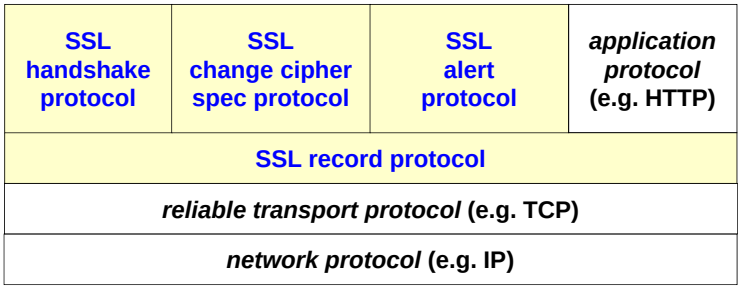


3. Assuming you have found a number of common parameters, the next step the server sends to the browser its certificate. It should be noted that the certificate must match exactly with the URL to which the user is trying to connect, so avoid shadow fake server or server. Upon receiving the public key the browser performs a series of operations equivalent to an exchange of challenge / response.

4. Optionally, the server may require authentication of the client, and in this case the browser will send the certificate to the user.
   Even in this case, this phase is followed by a series of operations equivalent to a challenge / response in the browser should demonstrate to control the private key corresponding to the public key contained in the certificate.

Once all these steps have been done correctly you will have a secure channel to which you can transfer all the desired data, and provides protection against possible attacks from the outside.

7.3.3 Architecture of SSL-3



From an architectural point of view SSL-3 is based on a network protocol (eg. IP) and to function properly it needs a reliable transport protocol (TCP, and then this means, but are not tied to use this protocol) .

TCP is going to create a SSL record protocol, which is a protocol for transmitting individual records SSL. These records can be used directly by application protocols; for example a HTTP transmission takes a HTTP payload and inserts it into an SSL record, which will end in a TCP segment that finally end in a series of IP packets.

The SSL record protocol is also used in the opening phase of the channel to make the handshake protocol, namely the stage of initial exchange during which the server and the client authenticate and negotiate their safety.

In the case where the channel to remain open for a long time is good practice to periodically update the algorithms and associated keys. For this purpose it has a specific protocol called *SSL change cipher spec protocol* that allows you to change the specifications of algorithms and / or keys.

Finally l'*SSL alert protocol* launches alarm if, for example, a package was duplicated, the MAC was wrong and so on. Obviously these types of messages must in turn be protected, and then they too are encapsulated in an SSL record protocol which provides the basic protections.

7.3.4 Session-ID

Consider the typical web transaction, in which a user connects to a page, going to open a normal HTTP channel. The operations that are carried out are as follows:

1. open
2. GET page.htm
3. page.htm
4. close

At this point the browser begins to interpret the page, and see that the page requires loading of other resources. Therefore it will be necessary to go back to open the channel with the server:

1. open
2. GET home.gif
3. home.gif
4. close

1. open
2. GET home.mid
3. music.mid
4. close

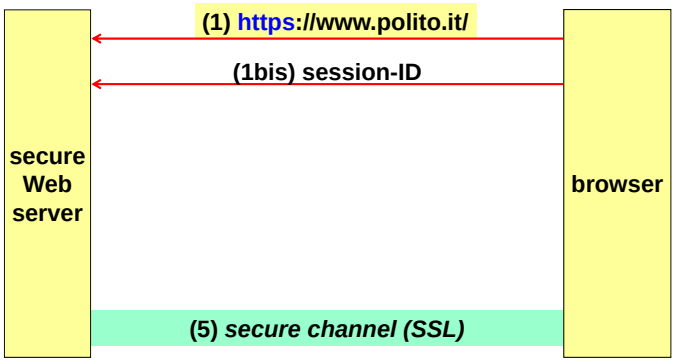
If instead of using a normal HTTP channel is used a secure SSL channel, this method would be unmanageable because if every time you need to download a resource you must go to renegotiate the parameters for SSL encryption, the connection is much more unwieldy.

To avoid re-negotiate each time you connect the cryptographic parameters, the SSL server can offer a session identifier, the so-called *session-id*. This means that more connections can be part of a same logical session (hence the fact that SSL implements a secure session level rather than individual secure channels).

If the client has a connection to the opening of the session-id valid, you skip the negotiation phase and proceed immediately with the dialogue SSL, then going to jump across the stage handshake.

Note that the use of session-id is not mandatory. The server can also refuse to use (absolute, or after a certain period of time from their issue).

SSL session-id



The user wants to connect to a server, and together with the connection request to the server also sends the session-id that has been assigned in a previous connection.

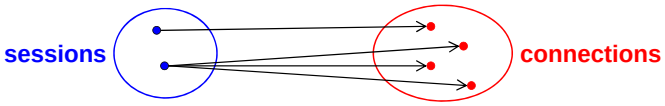
The server verifies that this session-id is correct, and if so activates the secure channel on which you can immediately begin transferring data.

The server realizes that if he is trying to use some session-id it is also the actual owner, as to that identifier will be associated with a set of security parameters. If a user then tries to use a session-id not belonging to it will not know what parameters go to use, and then the server is not going to open the secure channel.

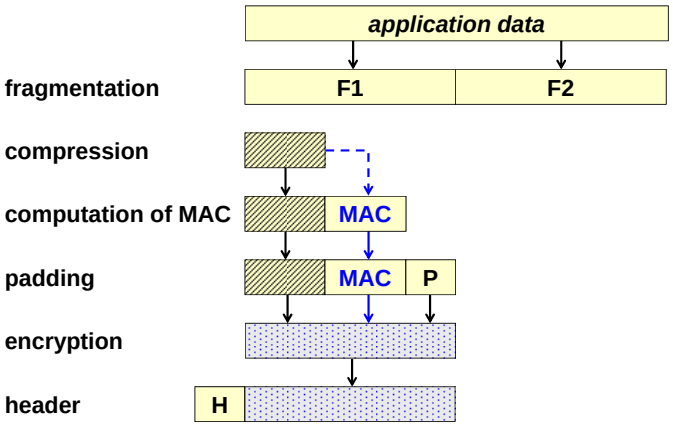
SSL / TLS: Sessions and connections

A SSL session is an association between client and server logic. Is created by the handshake protocol and defines a set of cryptographic parameters. A session is used at least, a link or SSL connection, but potentially many (in case you had activated the session-id).

A *SSL* instead is a temporary SSL channel between client and server, thus corresponding to a TCP channel, and is associated with a specific SSL session. It therefore has a 1: 1 correspondence.



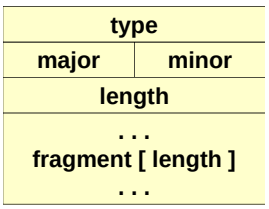
7.3.5 SSL-3 - TLS record protocol



Suppose you have application data that must be transported within an SSL channel.

1. The first step that performs SSL is to split the data into multiple fragments, although these data are part of a some stream.
2. Before applying to each of these fragments security properties typical protocol is made a transaction (optional) compression. This process reduces the amount of data to be transferred but requires greater computational power on both the client and the server. In the case of security, however, the compression has another big advantage: because compression tends to eliminate duplicate parts and replace them with alphabets common, this reduces the information available for a possible attacker were to try to attack the cipher text. In general, so it is always a good idea to do data compression before these encryption; do the compression after encryption is not only useless but will be harmful in terms of calculations.
3. Then the fragment (original or compressed) is used to calculate the MAC. In this way it provides integrity and authentication to single fragment.
4. Assuming that you are using a block algorithm, you must do the padding so as to reach the fragment size multiple of the basic block. This operation is obviously not necessary in the case where it was decided to perform an encryption of stream type.
5. At this point, the fragment is ready to be encrypted with the algorithm chosen, thus going to also provide confidentiality.
6. Finally the Accordingly a small fragment is encrypted SSL header.

TLS-1.0 - Record format



- *Type*: allows you to specify what data is transported, which may be of higher level protocols or simply application data.
- *Major & Minor*: identify the major and the minor number of TLS. Note that for TLS-1.0 is not written major = 1 = 0 and less, but is written major = 3 and lower = 1; TLS because this is seen as the continuation of SSL, TLS and began to exist as of version 3.1 of SSL.
- Length: 16-bit field that identifies the fragment length. If this value is leq 214 indicates that the records are not compressed and this encoding is maintained for compatibility with SSL-2; records compressed instead have a value of leq 214 + 1024.

TLS - Calculation of MAC

= MAC message _digest (key, seq _number type || || || fragment length)

As for the calculation of the MAC, it is calculating with a message _digest that depends on the algorithm that was negotiated between the client and the server in the handshake.

The digest is calculated from the key shared communication; that key will be different depending on whether the communication goes from the server to the client or the client to the server. These keys are known to both peers but are not interchangeable.

The digest is calculated not only on the data of the fragment itself, but also on its length (to prevent someone make attacks of extension), the protocol version (to prevent attacks that seek to transform a package in a package of a'other version), type (to protect the integrity of the fragment transmitted) and also the sequnce number (to protect against replay attacks).

Note that the sequence number, an integer from 64 bits, it is not explicitly written anywhere; its value is implicitly known by both peers as they count all the fragments.

In general 64 bits are sufficient to protect against any type of attack. This is not true in case of SSL were used to create the tunnel that remain active for a long time; in this case, if you go to transfer more than 2^{64} packages you have an overflow, and for that reason to avoid generate yourself a replay attack should close and reopen the channel.

7.3.6 Problems SSL-2

For some years SSL-2 has been deprecated because it was characterized by a number of security issues:

1. SSL 2 was equipped with a version of export, and in this version it had a reduction to 40 bits of all the symmetric keys, both the encryption is the one used for the calculation of the MAC. Regulation of export provided for a reduction in 40-bit only for the keys used for encryption, but did not impose any constraint for authentication keys; consequently they had no reason for a reduction in strength of the authentication process, and not just the data were spiabili but were also falsifiable.

2. The calculation of the MAC was done with a keyed-digest algorithm custom, which was not very safe .

3. The bytes of padding, used for encryption in the case of block algorithms, are included in the calculation of the MAC but the length of the padding is not. This allows for an active attacker to erase bytes at the end of messages.

4. Cyphersuite rollback attack: because the phase handshake was not authenticated, an active attacker could change cyphersuite in the Hello message to force the' use of weak encryption algorithms. In any case, the encryption keys must be based on at least 40 bits because the encryption is mandatory in SSL-2 (if the same attack would be possible also in SSL-3 would be a disaster because in this new version can be removed totally encryption).

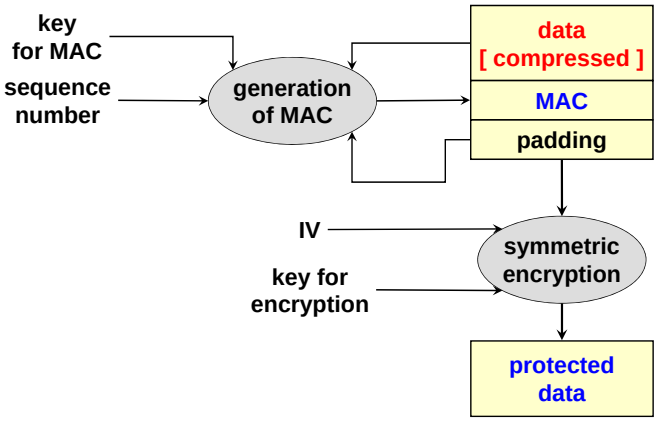
7.3.7 SSL-3 - Troubleshooting SSL-2

- The limitations regarding the export concern only the encryption keys, which are reduced to 40 bits. Keys authentications are long much as was negotiated, typically 128 bits.
- For the calculation of the MAC using the HMAC, much stronger than the keyed digest custom used in SSL-2.
- The padding is included in the calculation of the MAC.
- The handshake is authenticated (except for posts by Change Cipher Spec) indirectly through messages Finished. What is done is to perform fully the handshake process, and once negotiated algorithms and keys in the last message is a calculated value that is used to authenticate all the previous exchange. Authentication therefore not about individual messages but the entire handshake procedure.

SSL-3 - News compared to SSL-2

- Optional compression of data, to be made prior to encryption.
- Data encryption is made optional, so you have only authentication and integrity .
- Adding a protocol that provides the opportunity to renegotiate the connection. This allows you to periodically change the keys and algorithms.

7.3.8 Privacy



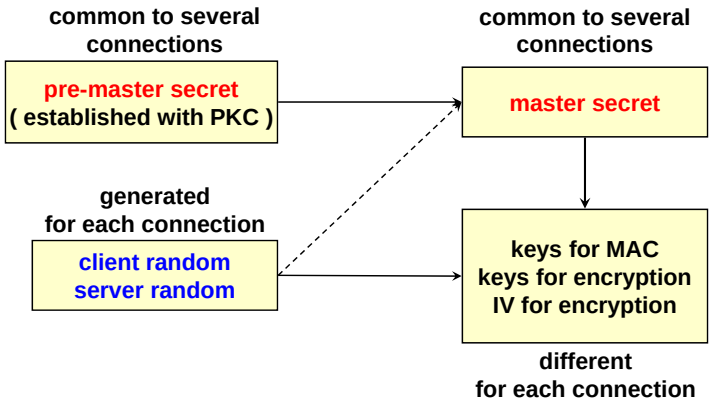
Regarding data protection, we want to highlight now what relationship exists between the different keys that are present.

First you have the data, possibly compressed, that you want to encrypt. Knowing how great will be the MAC, you can go to calculate the size of the padding automatically.

The data and the padding, together with the key of the MAC and the sequence number, are provided as input to the algorithm that must generate the MAC (negotiated during the handshake) which will be inserted in the fragment.

At this point, supposing to have a symmetric algorithm block is made symmetric encryption of the fragment through the use of an initialization vector (also implicit, both peers are aware of them) and the encryption key.

7.3.9 The relationship between keys and sessions



First, the first time that two peers make contact with SSL, set via a so-called public key encryption *pre-master secret*; this will be common to multiple connections.

At this point you can generate a master secret, which is also unique for the session, and therefore common to multiple connections.

Whenever you connect instead, and then for each connection, generates two random numbers: *random* client and *random* server. These two numbers are used along with the pre-master secret in the first free session to generate the master secret, but most are very useful in subsequent times it is used together with the master secret using special algorithms for the calculation of the keys to the MAC, keys for symmetric encryption and the initialization vector; all these data will therefore be different for each connection.

7.3.10 Mechanisms “ephemeral”

One of the peculiarities of SSL over other protocols is the possibility of using mechanisms called ephemeral. These are relevant when dealing with large amounts of data or many connections to the same server.

An SSL server typically has a public key to prove their identity. The danger in using many times the same public key is that by dint of making signatures the attacker can infer something about the corresponding private key. To avoid this thing the server can then create the keys using throw-away asymmetrical generated on the fly. But since these keys generated on the fly, they will not be certified. What is then done is going to sign these keys generated on the fly by using a public key certificate. The server must then be provided with a certificate not to use it directly against the client, but the client to communicate the public key that has been selected for a given connection.

Mechanisms for the exchange of keys you can use RSA or DH. The step of generating a pair of public / private key is extremely slow when using RSA, but very fast but if you use DH. For this reason, in the event that it is used RSA, typically the key pair is not used for a single connection but is reused several times.

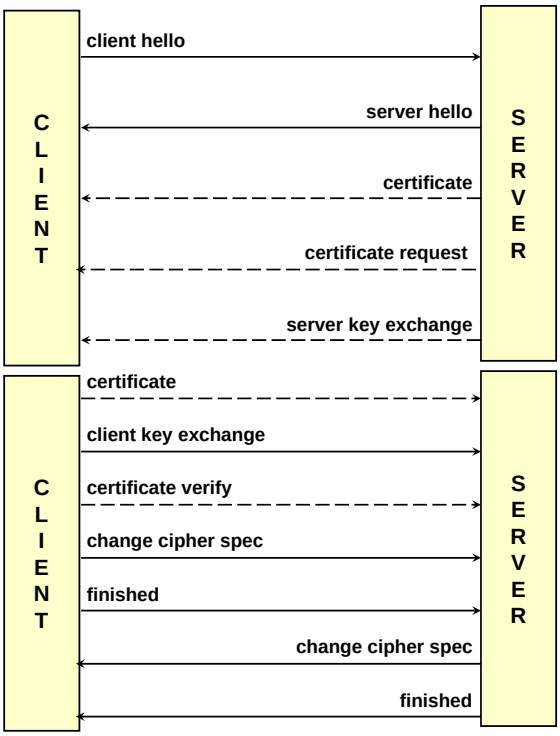
SSL also allows you to perform the so-called perfect forward secrecy. Suppose that at some point someone will be able to take possession of the private key that is installed on the server. If this person has registered all previous sessions, when he discovers that the private key can decrypt the packets not only future but also all gone.

If you have used a mechanism ephemeral, what is done is to use every time a pair of public / private key different for each connection. Therefore if someone could to attack the server's private key, could not decipher the traffic past but only the future, because each connection is protected by a different public key; the private key on the server is not used for privacy but only to sign the public key.

7.3.11 3-SSL handshake protocol

The handshake protocol is designed to:

- Agree on a pair of algorithms that are used for confidentiality and integrity.
- Exchanging random numbers between the client and the server serve as a basis for then generate their own encryption keys.
- Establish a symmetric key (which will then be derived from the encryption key and the authentication) via public-key operations. Are supported RSA, DH and Fortress.
- negotiate session-id.
- exchange all public key certificates needed.



In the figure you have all the packets that are exchanged during the handshake. The dotted line indicates that in some protocol implementation matching packets are optional.

The client initiates communication by sending a message called *client hello*, to which the server responds with a *server* hello.

After which are optional, because for example you decide to use the session-id, the exchange of the server certificate (*certified*), the certificate request to the client (certificate request), a message for the exchange the keys of the Serer (*server* key exchange), sending the certificate (*certified*) if the client is in possession of it, and the server has requested.

It is mandatory the client key exchange, which is used to generate keys specific for this connection.

Optional one more step to verify the certificate (*verify* certified).

Finally you have to send the message change cipher spec, which is used to activate the security protocols, and message *finished*, which serves to protect all the previous exchange. These last two messages are also sent from the server to the client to tell the client that he is going to turn on all the security measures and to protect the messages sent by him.

Client hello

The client hello serves to:

- Declare the SSL version preferred by the client.
- Send 28 bytes generated in a pseudo-random, the so-called client random.
- Send an identifier session, the so-called session-id. This field will have value of 0 if the client wants to start a new session, and will have a value other than 0 to ask to exhume a previous session. It is said that the server accepts values other than 0, may in fact have disabled support for session-id.
- List all cipher-suite client. This term refers to the set of encryption algorithms, key exchange and integrity that the client is able to process.
- (From SSL-3) List the compression methods supported by the client.

Server hello

hello The server is used to send:

- The SSL version chosen by the server.
- 28 pseudo-random bytes that constitute the random server.
- A session identifier, the session-id. If the client had entered 0 as a session-id, the server will use this field to send the client a new session-id, otherwise this field will have value equal to the session-id proposed by the client, but only if the server agrees to exhume the session . If the server does not accept, the server will send the client a new session-id.
- The cipher-suite chosen by the server; should be the strongest in common with the client.
- Compression method chosen by the server, including those proposed by the client.

Cipher suites

The cipher suite includes:

- The key exchange algorithm.
- The symmetric encryption algorithm.
- The hash algorithm (for MAC).

These are typically indicated by the strings, such as SSL _NULL _with _NULL _NULL (no algorithm is not used to exchange keys, no symmetric encryption algorithm and no hash algorithm). This in particular is the protection that is initially during the handshaking.

Certificate (Server)

During this phase, the server sends the client a certificate for server authentication. The certificate is of type X.509, and it is necessary that the content of the subject field or field subjectAllName conincida with the identity of the server (and therefore must contain the DNS name, IP address, ...).

The certificate can be active only on keyUsage signature or even the cipher. In the case in which the certificate is only for signature then it is then required in the step of the server-key exchange; This serves to indicate that the server will use mechanisms ephemeral.

Certificate request

Message sent from the server to the client, is used to authenticate the client during the exchange SSL.

In particular, the server asks the client to authenticate itself by sending an X.509 certificate. In addition, this message also contains a list of CAs that the server believes trust. This means that the browser when it receives this message looks at what the CA who issued a certificate for the user, and therefore seeks to have been issued by a CA that the server trusts. In the event that the user did not have a certificate issued by one of the trusted CA, the protocol will hang because the server does not trust the identity of the client.

Server key exchange

This step is only used if the server wants to use the mechanisms ephemeral, and contains the public key for the exchange server keys.

In particular, this message is only required in the following cases:

- The server has an RSA certificate just for signing up.

- You have chosen to use anonymous or ephemeral DH to establish the master-secret. (With DH anonymous means to exchange keys DH in which the keys will not be signed by the server. In this way, the client will have no certainty about the identity of the server, but you will still have the protection of the communication channel by third parties)
- There are problems of exports and then you have to use RSA / DH temporary.
- Required if using Fortress.

This is the only message that contains a signature explicit asymmetric server, because the key put in this exchange must somehow be authenticated.

Certificate (client)

If the server has requested a certificate to the client and that it has, is then sent the message certified client.

This message carries the certificate for client authentication. The certificate must have been issued by one of the trusted CA listed in the message from the server certificate request.

Client key exchange

The client provides the information needed to determine the encryption keys and MAC. There are several possibilities:

- The client can send the pre-master secret that he has invented, encrypted with the RSA public key of the server (temporarily, if one were to use mechanisms ephemeral, or certificate).
- Ability to send the public part of DH, in the event that it had been decided one key exchange based on DH.
- Enter the parameters of the algorithm Fortress.

Certificate verify

The phase certified verify is a phase in which it is made an explicit proof of signature.

The hash is calculated on all handshake messages that precede this, and is signed with the private key of the client.

This is only necessary in the case of client authentication, that is, only in the case where the server had requested the client to authenticate. In a sense this is the answer to the challenge from the client to the server.

Change cipher spec

The change cipher spec is one of two final messages. After this message all subsequent fragments are protected with the algorithms and keys that have been negotiated in the previous messages.

Strictly speaking, this message is part of a specific protocol and no handshake.

Analysis indicates how eleminabile, but for the moment has not been eliminated either in SSL or TLS versions later.

Finished

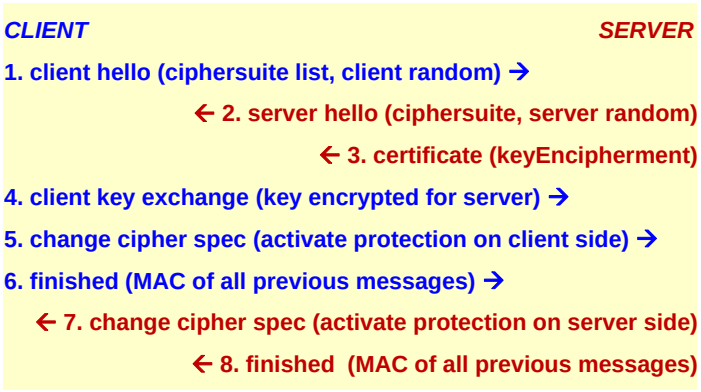
This is the first message that is protected with algorithms negotiations, and it is critical to authenticate any exchange because:

- contains a MAC computed over all previous handshake messages (except change cipher spec) through the use of the master secret.

- Evita attacks man-in-the-middle rollback, acts to do version downgrade or downgrade ciphersuite.
- You diferenziato for client and server for each of the two calculates the MAC messages that he sent.

7.3.12 TLS - Implementation Examples

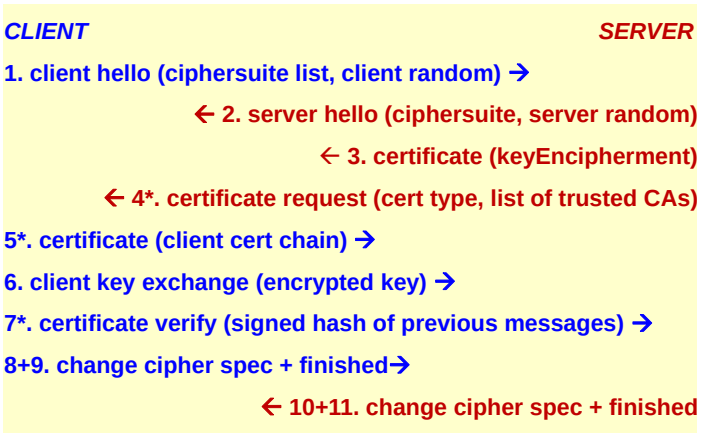
TLS, no ephemeral key, no client auth



This is the connection diagram easier there is, the one that minimizes the number of messages exchanged.

1. The client sends the *client hello*, in which the basic elements are the client and the random ciphersuite list.
2. The server responds with a *server* hello, where he chose a specific ciphersuite and sends its server random.
3. At this point, the server sends the message *certified*, where the important thing is that the certificate sent serves not only for digital signature but also for data encryption.
4. The client, using the client and the server random random, generates the master secret and sends this encrypted key to the server using the certificate sent to him by the server in the previous mesaggio.
5. The client can now activate all security measures on the client side by sending the message *change cipher spec*.
6. Enter the message *finished*, that contains the MAC of all the previous messages (except for the change cipher spec).
7. The server responds with its *change cipher spec*, with which states the intention to activate the safety measures from the server side the next message.
8. Enter the message *finished* that contains the MAC messages 2:03.

TLS, no key ephemeral client auth

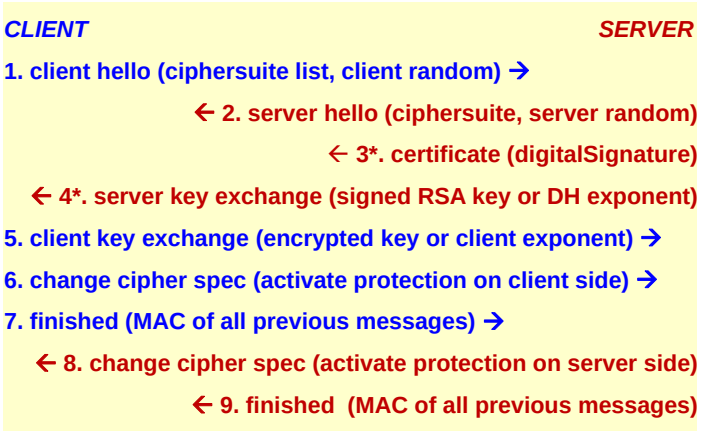


1. *Client hello.*
2. *Server hello.*
3. *Certificate (server).*
4. The server sends the client a messsaggio of certificate request containing the type of certificate and the client will have to have a list of CAs that the server trusts.
5. The client sends its own certificate in the form of client cert chain, because the server has indicated the root CA to which he trusts and consequently the client has to show how the CA that issued its certificate you can get to a root CA trusted.
6. In addition, the client sends a message of client key exchange, with which the client sends the key encrypted with the mechanisms of the server.
7. Phase *verify* certified, which is the only case of digital signature explicit by the browser. It performs the hash calculation of all the previous messages, signed with the private key of the client. This is done to prove possession of the private key corresponding to the certificate was sent to the step 5.

8+9. *Change cipher spec + finished.*

10+11. *Change cipher spec + finished.*

TLS, ephemeral key, no client auth

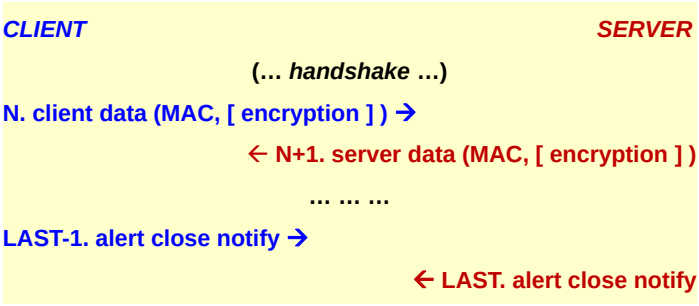


1. *Client hello.*
2. *Server hello.*
3. *Certificate (server)*, this time bearing the text digitalSignature: this means that the certificate of the server features can not be used to do encryption, but only to make digital signatures. The client then can not send to the server the secret encrypted which serves to derive the keys.
4. As a consequence the server chooses to generate the keys, such as an RSA key, or an exponent DH, and sends the public component to the client signed by the private key corresponding to the certificate sent to the step 3.
5. The client sends a random number generated by him, encrypted with the key sent to it from the server in step 4. If, however, the server had sent a representative DH, the client does not send a random but also sends him an exponent DH but unsigned. At this point you have to wonder how it is possible to be sure that it is the component DH chosen by the client and not something manipulated an external . . . The integrity of the exponent is guaranteed by the operation performed in step 7, when you go to calculate the MAC of all messages sent from the client! This provides a signature implied exponent.

6+7. *Change cipher spec + finished.*

8+9. *Change cipher spec + finished.*

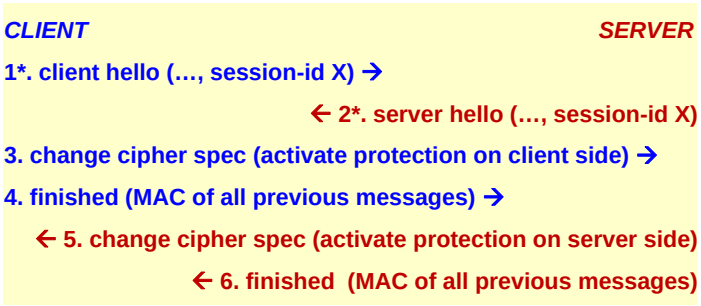
TLS, data exchange and closing channel



1. phase handshake successfully completed.
2. At N -th message the client sends its data to the MAC and optionally protected with encryption.
3. The server responds with the message $N+1$, which contains your data (also protected with MAC and optionally encrypted).
4. At some point you decide to close the channel. The message LAST -1 is of type *alert close notify*, part of Alert protocol, which is used to notify the server that the client is going to close the channel.
5. The server responds to the client message with a *alert close notify*, with which the server notifies that for him the channel was closed.

Note that both messages 4 and 5 are protected with a MAC, because otherwise a third might close the channel against the will of the two peers.

TLS, resumption of a session



1. In the phase of *client hello* between the different parameters is indicated a session-id X.
2. In *server hello*, in the case where the server to agree to continue a previous session, there will be the same value for the session-id.
3. At this point all the other phases of the handshake are skipped, because the server is aware of the parameters associated with the previously negotiated session-id X. It has thus sending the change cipher spec, which activates the client-side security.
4. message *finished* containing the MAC of the message 1.

5+6. *Change cipher spec* + *finished*.

7.3.13 Transport Layer Security (TLS)

Starting from version 3.1 SSL changed its name to *Transport Layer Security (TLS)*. This is due to the fact that Netscape, the author of the initial version of SSL, when he finished his business donated the IETF protocol. It is, however, had a very strong opposzione Microsoft could not accept that a protocol invented by a competitor was standardized with a name that would evoke the same competitor, the IETF has finally succumbed to such pressure and decided to use the name TLS.

The TLS-1.0 is documented in RFC-2246 to January 1999, and practically implements SSL-3.1 (corresponds to 99 % with SSL-3). The only changes that have been made are:

- Use of standard algorithms such as HMAC instead of keyed-digest custom invented by Netscape.
- Emphasis on algorithms that did not require payment of royalties. The ciphersuite basis using DH + DSA + 3DES, or TLS _DHE _DSS _with _3DES _EDE _CBC _SHA.

TLS 1.1 (SSL 3.2)

The next version of the protocol brings some improvements, in particular helps to protect against a series of attacks on CBC. In the first version of TLS was possible to create so-called cryptographic oracles in case it was used with a CBC padding wrong, that is going to change the bits of padding and providing them to the server the attacker can use the server to find out if he was wrong or not wrong to make changes; This coupler is based on the fact that the server provides the different responses depending on whether the MAC is wrong or decryption is wrong.

To avoid this kind of attacks the initialization vector implicit that was calculated and never transmitted is now replaced with an explicit initialization vector. Any errors in the padding using the generic message alert bad _record _MAC, which does not distinguish between errors in the padding and data errors; in the previous version of TLS is used the message decryption _failed, and then we went to deliver the information to the attacker if the attack had been made on the part of padding or on the data.

Minor enhancements include:

- IANA Allocation to the task of recording all the parameters of the protocol.
- The fact that a channel is closed prematurely no longer means that you can not resume the session.
- Assigning developers of a series of additional notes to clarify various possible attacks.

TLS 1.2 (SSL 3.3)

Version standardized in August 2008 with the RFC-5246.

Main changes:

- The ciphersuite has been extended to also specify the PRF (pseudo-random function), the functions that the client and server are using to generate the random numbers to be used for key generation.
- As SHA-1 had now been deprecated, begins to be used SHA-256 in e-Finished (so that it is part of the strong authentication handshake) and is made optional (but highly recommended) to use it in the MAC.
- Introduction of support for authenticated encryption based on the use of AES GCM mode or CCM.
- incorporates extensions (mechanisms to add additional parties to the Protocol) and all ciphersuite based AES that were described in a RFC apart.
- The default TLS ciphersuite becomes _RSA _with _AES _128 _CBC _SHA.
- become deprecated the ciphersuite based on IDEA and DES.

7.3.14 TLS and virtual servers

Within HTTP / 1.1 when a client connects to a server in addition to indicating the GET or POST in the resource that you wish to access, it is also mandatory to use a Host header that indicates the virtual host to which you want to connect. This is done because a single IP address (and therefore a physical machine) can match multiple virtual web servers.

This distinction is made easily in HTTP, but it is difficult to do with HTTPS because TLS is activated before HTTP, but the server must send the client certificate matches the name of the server to which the client will connect. However, the client will provide this information only within the HTTP protocol, and therefore the server having inside multiple certificates will not know which of these having to send to the client.

To solve this problem you can:

- Use of collective certificates (wildcard) in the event that the various virtual servers hosted all correspond to the same domain. It therefore has a single private key shared between all servers.
  However, this solution has the problem that the collective certificates are treated differently by different browsers.
- Create a certificate for the server that contains a blank field in the of subject and contains, as subjectAltName all the names of the various virtual servers hosted. Also in this case the private key is shared by all virtual servers.
  This solution has the problem of having to re-issue the certificate to any addition or deletion of a virtual server.
- Use the extension *Server Name Indication (SNI)*, documented in RFC-4366. In particular during ClientHello the client can send the SNI, containing the name of the virtual server to which the client will connect.
  The problem with this solution is that it is an extension of his support is not mandatory, and therefore do not is used by all browsers and servers.

7.3.15 Datagram Transport Layer Security (DTLS)

Given the great success of TLS was proposed DTLS protocol, which is nothing but the application of the concepts of TLS transport datagrams (and therefore potentially provide protection to UDP).

However as part of the protections afforded by TLS are possible due to the fact that its use is associated with the use of a reliable transport protocol, being a UDP transport protocol is not reliable will not be possible to have all the security services offered.

For this reason DTLS is not supported by all, or at least is in competition with other solutions. For example is in competition with IPsec and with the possibility of implementing security directly at the application level.

For example, in the case of protection of the SIP protocol you can use:

- IPsec, and then the problem of properly configure the client and the server.
- TLS, if you decide to use SIP _over _tcp.
- DTLS, if we were to use SIP _over _udp.
- Secure SIP, where security is implemented directly within the SIP protocol.

7.3.16 Heartbleed

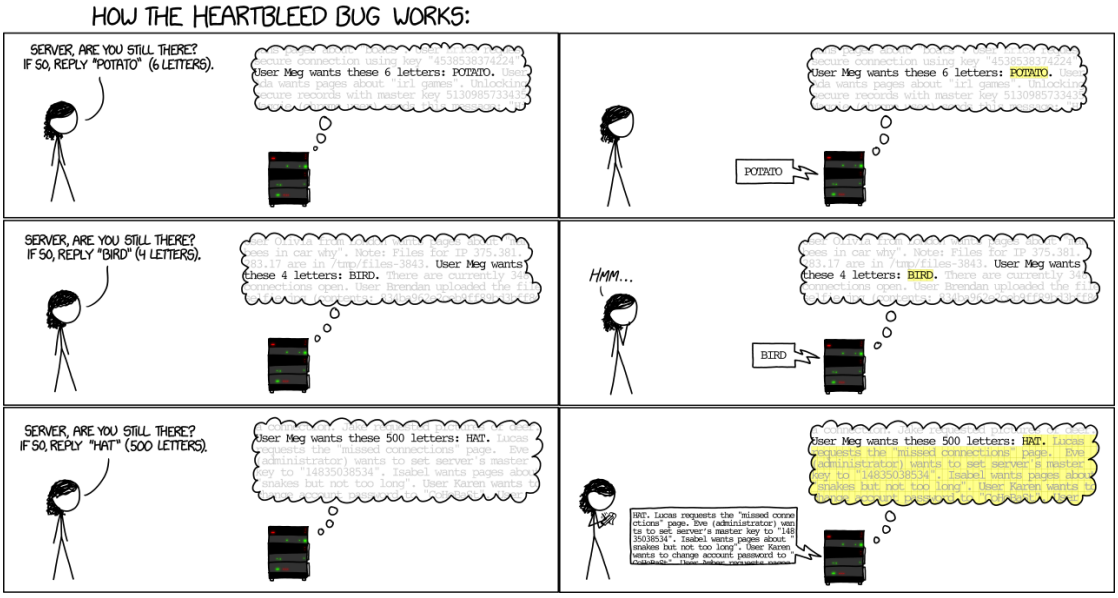
Heartbleed is an attack discovered in April of 2014.

Everything starts from RFC-6520, which inserts an extension called Heartbeat. This concept is also present inside other network protocols, and consists in sending periodically packets between two peers when they are not exchanging any data, with the sole purpose of not to terminate the connection. TLS in this extension is used to keep a connection open so you do not have to constantly renegotiate the parameters. This is very important for SSL, but it is absolutely vital to DTLS because in this case there is the concept of the session, and then runs the risk of having to renegotiate each time from the beginning all the parameters.

Also this extension is also useful in PMTU discovery, ie are sent heartbeat messages to find out what is the best path MTU between the client and the server.

The safety issue related to this extension is documented in CVE-2014-0160 (CVE = Common Vulnerability Exposion, which is a global DB indicates that all known vulnerabilities. At each new vulnerability is associated with a number dialed by the year and the number in that year), and not generally concerns the TLS but regards the specific implementation within openssl. In particular, it has a problem of buffer over-read, ie the TLS server responds with more data (up to 64K) of those sent in the request heartbeat.

As a consequence we have that the attacker can obtain sensitive data in that block of RAM, such as user + pwd or even the server private key is not used in case an HSM.



7.4 Security in HTTP

SSL is the leading security protocol used on the Internet, but not the only one. In fact there are other protocols that apply especially when you go to use Web applications.

In general the term Web application refers to applications that carry their data via the HTTP protocol level 7.

Currently there are two versions in use of the HTTP protocol, ie HTTP / 1.0 and HTTP / 1.1. Inside the RFC that defines the first version of the protocol are defined two very basic security mechanisms:

- Address-based: the HTTP server controls access to its resources according to'IP address of the client.
- Password-based (Basic Authentication Scheme): determines the access to a resource is limited by a username and password, which are transmitted encoded in Base64. However this is not an encryption algorithm, and therefore serves only to perform a sort of blurring of the data.

Both schemes turn out to be highly insecure, and this is already recognized by the authors who have written that HTTP RFC supposed that its mechanisms are used within a secure channel.

HTTP / 1.1 introduces a new security mechanism based on symmetric challenge called digest authentication.

Currently all of these security mechanisms are summarized in RFC-2617 that speaks of authentication at the HTTP level.

7.4.1 HTTP - Basic authentication scheme

```
GET /path/alla/pagina/protetta HTTP/1.0
  HTTP/1.0 401 Unauthorized - authentication failed
  WWW-Authenticate: Basic realm="RealmName"
Authorization: Basic B64_encoded_username_password
  HTTP/1.0 200 OK
  Server: NCSA/1.3
  Content-type: text/html
  <HTML> pagina protetta ... </HTML>
```

After opening an HTTP channel, the client requests a resource using for example as GET. From a point of view of the client protected pages are indistinguishable from those not protected (look at the path of the example of the resource, the client for this is not at all significant).

In response to this request, the client will receive an error message HTTP / 1.0 401, unauthorized access; this error is level 4, which is a temporary error. It is classified in this way because the client had no way of knowing that it was necessary to authenticate to gain access to that resource. As a consequence of the HTTP channel is not closed, but is left open in the event that the client wants to authenticate.

For this purpose, the server notifies the client via the WWW-Authenticate header that to have access to the resource authentication is required for a basic realm. Note that this realm is not to be confused with the Kerberos realm. This is just a suggestion that the server gives the client to allow him to remember which username and password to use. You would then have realm = "Portal of the teaching of the Politecnico di Torino".

When the browser receives this header typically shows a popup automatic (not written by the programmer of the web page) where it says something like "The server requested authentication for the realm ..." and asks to enter username and password.

The user will then enter a username and password in this popup, and then it will be the task of the browser to go to transmit the channel remained open through the use HTTP Authorization header. In this header you have an indication that you are trying to perform basic authorization, followed dall'username and password encoded in Base64. The security level of this mechanism is very low, because chunque interceptions this header can safely read username and password simply by decoding the data.

Assuming that no interceptions this communication, the server will receive the Authorization header and it will do the decoding. If user / pwd are valid and above are allowed to have access to the resource (authentication and authorization are performed in a single step), then the server will respond to the original request with a message HTTP / 1.0 200 to be followed by the requested resource.

7.4.2 HTTP - Digest authentication

The digest authentication is an improvement of the basic mechanism, which allows you to make secure authentication even outside of an SSL channel.

This mechanism was introduced in RFC-2069, now technically obsolete, but is considered as the base case in RFC-2617.

What is done in practice is to calculate a keyed-digest based on the user's password:

$$\text{HA1} = \text{md5 (A1)} = \text{md5 (}\textit{user}\text{ ":" }\textit{realm}\text{ ":" }\textit{pwd}\text{)}$$
$$\text{HA2} = \text{md5 (A2)} = \text{md5 (}\textit{method}\text{ ":" }\textit{URI}\text{)}$$
$$\text{response} = \text{md5 (HA1 ":" nonce ":" HA2)}$$

The nonce, sent from the server to the client, is used to prevent replay attacks.

The authentication server can also insert a field "opaque", ie a binary string not intelligible by the browser, to carry any state information (eg. SAML token).

Digest authentication scheme

```
GET /private/index.html HTTP/1.1
  HTTP/1.0 401 Unauthorized - authentication failed
  WWW-Authenticate: Digest realm="POLITO",
    nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
    opaque="5ccc069c403ebaf9f0171e9517f40e41"
Authorization: Digest username="lioy",
  realm="POLITO",
  nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
  uri="/private/index.html",
  response="32a8177578c39b4a5080607453865edf",
  opaque="5ccc069c403ebaf9f0171e9517f40e41"   pwd=antonio
  HTTP/1.1 200 OK
  Server: NCSA/1.3
  Content-type: text/html
  <HTML> pagina protetta ... </HTML>
```

The scheme begins with a GET on the client side of a secure page precise. This request generates the error message HTTP / 1.1 401, following the header WWW-Athenticate with which the server indicates that it takes an authentication type Digest, for the realm = "Polito". Inside the header are also present the nonce, which will be used for the calculation of the digest, and the opaque, that the client should return it unchanged.

Upon receipt of this message, your browser prompts the user to enter a username and password, and based on these credentials will build the answer, that will be sent to the server via the Authorization header. If the digest will be calculated correctly and the opaque will be equal to the original, the server will respond in the affirmative with HTTP / 1.1 200 enclosing the requested resource.

7.4.3 HTTP and SSL / TLS

HTTP is often used in conjunction with SSL / TLS. You have two alternatives:

- *TLS* then HTTP, or before you open a secure channel and successively HTTP is used for data transport.
  (SSL then HTTP is used but it was not never documented because SSL was initially a proprietary protocol Netscape)
- *HTTP* then TLS. In this case it has the initial opening of a normal HTTP channel, and the server will decide if you want to make safe or not. It therefore has the opportunity to upgrade to TLS channel of a normal HTTP / 1.1.

The two alternatives are not equivalent, mainly because they have a different impact on applications, the firewall and the IDS.

- In an application based on TLS then HTTP security is not managed by the programmer of the Web server, but is managed by the system: it is the duty of the latter to enable TLS first to make available the Web application. This has the advantage of requiring less work for developers, but at the same time they will have no control over the security systems; need to trust the work of systems. It is therefore necessary to have a synchronization between the work of programmers and systems analysts that of.
  In-based application instead of HTTP then TLS is the developer who has to take care of security: when a user requests access to the protected resources necessary cease normal dialogue HTTP, transform the channel safely and only then will resume dialogue HTTP. This results in an increased workload for the developer as it should also take care of security,

but this can also be seen as an advantage as it is the same developer which controls if and what security you want to have.

- Concerning firewalls, the two solutions are not equivalent mainly for the reason that they use different ports. If you are using TLS then you will have the HTTP port 80 for HTTP is not secure, and port 443 for TLS on which you go to vehicular HTTP. Consequently, the firewall is able to clearly distinguish from the non-secure HTTP fail.
  With HTTP then TLS instead there is only one channel, TCP 80, which at the discretion of the developer can be rendered safe for the transmission of certain resources. Consequently the firewall, relying exclusively on the doors, no longer able to make a distinction between secure and non-secure channels. It would be necessary to make such an exception content.

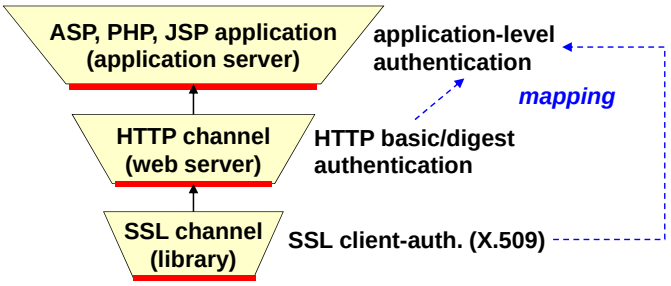
- As for IDS, if you are using TLS HTTP then the IDS will only be able to see a request to open a secure channel. Once the channel is open you can no longer see anything.
  With HTTP TLS instead then the IDS can see that the channel has been created and that a specific request to the server has decided to transform the channel secure. The IDS will however not be able to see anything as long as TLS will be active, but at least you have a control over what URL you activate security.

Note that these concepts are not only valid if you are using HTTP, but in general are valid for all protocols of level 7 TCP-based, and therefore can be based on TLS.

7.4.4 Client SSL authentication at the application level

Through the client authentication is possible to identify the user who opened a channel without going to require you username and password, but based on the use of a certificate.

Some Web servers allow you to do a mapping (semi-) automatically between the credentials that have been extracted from the X.509 certificate used to level SSL and HTTP service users and / or OS.



Authentication of a user can be done in three different ways:

- If the SSL channel was activated with client authentication, to authenticate the user must provide an X.509 certificate, in addition to having correctly answer the challenge.
  This mechanism if the server can perform a mapping application developer to know the credentials of the user who requested access to a particular resource.

- If you decide to use only SSL with server authentication, the next point where you have the opportunity to identify the user is at the HTTP level, where you can take a basic or digest authentication. Again the authentication result can be mapped.

- If you decide to use any of his previous solutions, the last option is that in the Web page, and then within the code, there is a form within which it is going to ask the user to enter a username and password.

These three solutions are certainly equivalent as regards the final result, as in any case the Web application will know the identity of the user who has opened the channel, but are not at all equivalent from the point of view of safety.

Here comes a very important concept nowadays called attack surface. Since the code is the weakest point of any system, you must try to expose the least amount of software to the attacker because the more you expose is more likely to contain bugs that can be exploited to execute an attack:

- If you level security SSL striker will go to exploit only bugs contained within that library; therefore you will have a very small attack surface.
- If you choose to do with SSL server authentication not only the attacker can exploit bugs contained within the SSL library, but will go to take advantage of bugs content within the Web server. Therefore the attack surface will be grown.
- Make safety top level allows the attacker to exploit all the bugs contained in the SSL library, the Web server and the page code implemented by the developer application .

What you try to do so is to do authentication on the lowest possible level. Unfortunately this is very often not possible because users rarely have an X.509 certificate, or even if they had not told us that the system administrator has enabled the function of mapping credenzialità. As for the authentication at the HTTP level, very often the basic or digest authentication are not activated by the application developers (especially for ignorance). So in general, authentication is performed by using the form inserted directly into Web pages, because this is the most comfortable solution for application developers.

username and password in a form?

Technically not important security contained in the page where you enter the data, because the actual security depends on the URI of the method used to send your username and password. This means that the page can also be transmitted using HTTP (http://www.ecomm.it/login.html), but the user's credentials have to be sent using HTTPS (`<form ... action = "https://www.ecomm.it/login.asp" >`).

However proceed in this way is a mistake. From a psychological point of view it is important to the security of the page where you input the data because few users have the technical knowledge required to verify the URI of the method used for sending.

7.5 payment systems

The electronic payment systems are increasingly important because now many purchase transactions are made through Web channels.

In the past there was a tentantivo a Dutch professor of developing the perfect electronic money (from a scientific point of view), called DigiCash, but this system was a failure due to technical problems and regulatory. Governments have always been the terror of electronic money as well as money also electronic money is not traceable, and above all there is the risk that someone creates the virtual currency without having the corresponding real money, with a high risk of inflation.

Today all governments prefer that payments are made electronically through the use of credit cards, because this allows you to have perfect control of the citizen knowing what to buy and at what time.

The method of payment by credit card is via the transmission of the card number on a SSL channel; the use of a secure channel ensures the protection of the transmission of the card number during transit over the network, but does not provide any guarantee against fraud. A few years ago VISA Europe said that 50 % of fraud attempts stem from Internet transactions, but were only 2 % of its turnover.

Fraud may occur for example when you are at the restaurant, and when you pay the bill provides its own credit card to the waiter. This, if not controlled, could copy the numbers on the card, and could then use it to make purchases on the Internet. Internet thus becomes one of the

principal means on which occur fraud. In the past if you clonava card needed to go to a store to shop, but now you can buy anything quietly staying at home.

7.5.1 Security of transactions based on credit card

Over the years there have been several attempts by the major credit card providers d of creating safer solutions. Specifically:

- *Secure Transaction Technology (STT)*, born of a collaboration with Microsoft VISA.
- *Secure Electronic Payment Protocol (SEPP)*, created by Mastercard, IBM, Netscape and others.
- *Secure Electronic Transaction (SET)*, born from the merger of STT with SEPP.

Secure Electronic Transaction (SET)

SET is not a payment system, but is a set of protocols for use in open network safely existing infrastructure of payments by credit card.

It is characterized by two main features:

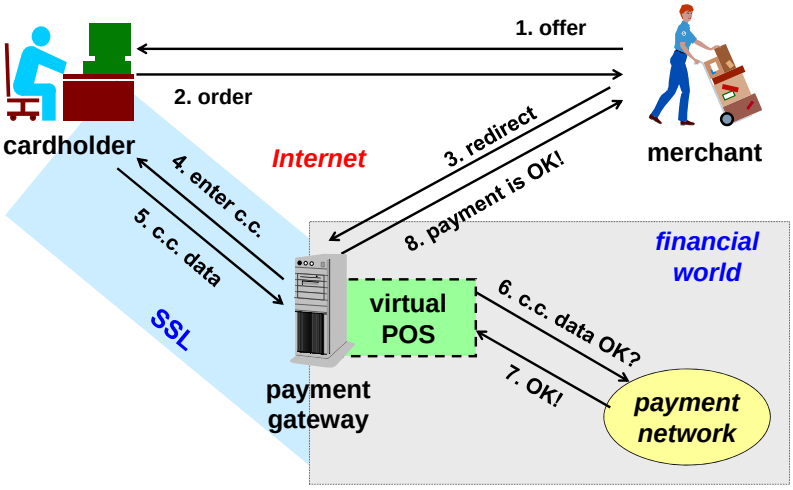
- Use X.509v3 certificates dedicated exclusively to transactions SET in order to ensure the safety of the various elements. These certificates were very expensive, and has been from the outset an obstacle for those who want to start making sales via the Internet.
- SET been designed in such a way as to ensure the privacy of users because it shows each party only data that compete.

Technically SET standard very valid, but it has two serious design errors:

- Put a very high price of admission for the purchase of an X.509v3 certificate.
- Relying on the fact that the user should have on his PC a so-called *set* wallet, which is software that contained public keys, private keys, payments and so on. This is considered a failure because electronic payments must absolutely be based on using a browser, and does not force users to make purchases through software; the latter also has a management problem, because all the problems that users will have to be resolved by the developers of this software.

All these features have led to the failure of this standard, which is now almost obsolete.

Architecture payment via Web



First, you can see the financial world (in gray), separated from the rest of the Internet. The problem lies in understanding on how to interface the payment methods that already exist with the Internet.

1. The merchant offers a range of products through its online sales channel, payable by credit card. After consulting the product catalog, the cardholder decides to make a purchase. At this point the merchant sends the cardholder formally a'*offer*.

2. The cardholder responds by accepting the offer. At this point, some merchants require the user to enter data in a form of credit card, and after performing all the necessary checks you can proceed to the payment itself. This is something perisolosissima, because it means that the data of the credit card are processed directly by the merchant, which if it is not a trustworthy person could use the data provided by the user to perform a fraud.

3. In general, the most common solution for online payments is to use a payment gateway. The merchant does not directly manage the data of the credit card, but when the request arrives Purchases made a redirects to a financial circuit with which it has entered into an agreement. The payment is not done then the merchant, but is done in the financial circuit.

4. Upon receipt of a request for payment, the first operation performed by the payment gateway is to open an SSL channel with the cardholder. After creating a secure channel asking the user to enter the data of the credit card.

5. The cardholder as requested undertakes to provide the data of his credit card.

6. At this point the payment gateway performs its function as a gateway, that provides a link between the world of the Internet and the financial world. In particular, the active gateway software called virtual POS, used to transform the data received in a transaction suitable to the financial world.

7. The normal payment network, after making all the necessary checks, it responds with a positive message.

8. At this point the gateway will respond to the merchant telling the payment has been successful, and then the order can considered himself concluded.

Compared to SET with this scheme does not have a privacy protection as the payment gateway will know whether the goods that you are looking to buy, both the data of the credit card. This chema but provides protection to the merchant, who no longer needs to know the data of the credit card of the cardholder but only has information regarding the goods purchased.

This scheme is based on the assumption that the buyer has a credit card, and that it is using a browser with SSL. Even if you are using a secure channel, the actual level of security depends on the configuration of both the server and the client.

Payment Card Industry Data Security Standard (PCI DSS)

PCI DSS is the standard for data security of the payment card industry, ie all entities that use payment cards.

Today it is a mandatory standard required by all of the financial institutions to allow the use of their cards for the purpose of Internet transactions.

This standard is much more prescriptive (ie from the very precise rules) compared to other security rules.

To be defined PCI DSS compliant must meet a set of requirements:

- *Build and maintain a secure network*

    1. Install and maintain a firewall configuration to protect cardholder data.
    2. Do not use defaults for system passwords and other security parameters from suppliers.

- *Protect data cardholder*

    3. Protect data stored cardholder.
    4. Encrypt cardholder data when transmitted over open, public networks.

- *Observe a program for the management of vulnerabilities*

    5. Use and regularly update anti-virus.
    6. Develop and maintain secure systems and applications.

- *Implement strong access control measures*

    7. Restrict access to cardholder data only if actually essential for the performance of the business.
    8. Give a unique ID to each user of the information system.
    9. Limit the possibility of physical access to cardholder data

- *Monitor and Test Networks regularly*

    10. Monitor and track all accesses network resources and data cardholder.
    11. Regularly test security systems and processes.

- *Adopt a Security Policy*

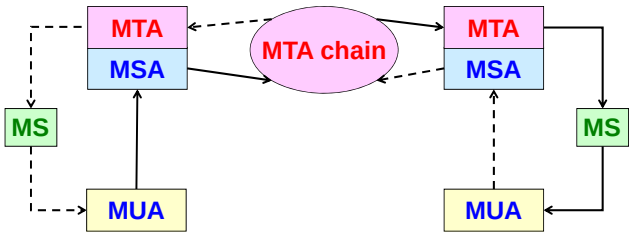
    12. Adopting a Security Policy.

Chapter 8

E-mail security

The email has the distinction of being the only service of wide use in which communication is not really end-to-end. In particular, e-mail works through the so-called *Message Handling System (MHS)*, where 4 IPs of different actors come into play:

- *Message User Agent (MUA)* .
- *Message Submission Agent (MSA).*
- *Message Transfer Agent (MTA).*
- *Message Store (MS).*



The moment you want to send the mail you are actually using an MUA, which allows you to compose your email message.

For shipping the MUA is based on a server called MSA, ie the one who knows how the email service worldwide and is able to introduce the message for transport.

The message is transported by a chain of MTA, or of special servers that are able to transfer mail from one part of the internet network. The last MTA output from the chain will be the one closest to the server where the mail is deposited by the recipient; this server takes the name of the MS.

In turn, the recipient will use a MUA to compose the message but to read messages that are received.

Note that the system works in both directions.

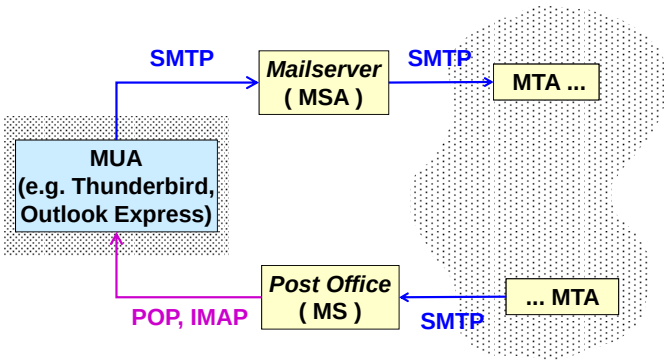
This system is similar to how the mail is delivered physics: the MUA is nothing more than the paper on which is written the text, the MSA is the mailbox where a letter is deposited, the MTA are the various postmen and means of transport with which the letter is transferred, the MS is the mailbox of the recipient and the MUA will be for example the key with which the recipient opens the box to take the letter.

From this diagram it can be seen that between the MUA of the sender and the recipient, there is no direct connection, but this makes sense given that the email service is an asynchronous service in which the various steps can be taken at different times. This feature is a big difference compared to other security systems seen so far, that instead they work in real time.

8.1 Mailbox - Charts implementative

8.1.1 E-mail client-server

The scheme client-server is the most classic and safe, even if today is no longer much in use.



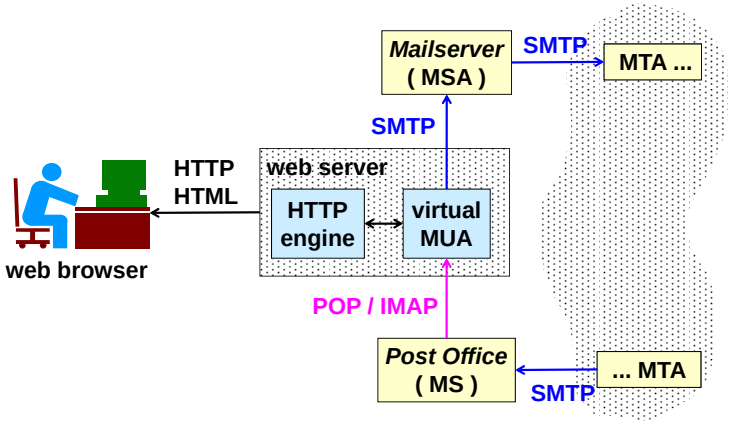
The user, equipped with a personal mail such as a PC, has installed on it a software MUA. This must be configured correctly so you know who is the Mailserver (MSA), to be able to send mail properly, and Post Office (MS), to read incoming mail.

Usually in software MUA not using the nomenclature MSA and MS, are concepts too technical. It is more likely that the MSA is listed as the Outgoing mail server and MS is referred to as Incoming mail server.

Sending and reading mail are based on the use of different protocols: Regarding sending using a protocol called SMTP push, which "pushes" the mail from one node to another network until it reaches MTA destination. For reading messages is used instead of the pull-type protocols, which allow you to "withdraw" the mail inserted inside your mailbox; the protocols used for this purpose are the POP and IMAP.

8.1.2 Webmail

Because very often users do not have their own personal device, are increasingly being used webmail. A webmail is something that has a web interface for e-mail system.



The user instead of having a real MUA on your personal device instead has a MUA virtual hosted by his e-mail provider. To interact with the MUA is exploited an HTTP engine, which allows you to use a browser to do the operations of shipping and receiving mail.

This way you are independent of the particular device, meaning that you can use many browsers to access the own email account. Furthermore, the user no longer has to worry about

properly configure the MSA and the MS, they will be configured appropriately by the virtual gesteri MUA.

From a point of view of security, when the user reads mail this is removed from the Post Office and is deposited on the servers of the provider of the electronic mail. In other words the mail is not the user's own, but it is the one who provides the service. In the above diagram once the user has removed the mail from the Post Office, the mail resides locally on the user's personal device; in other words it is totally owned by him.

The protocols used for transmission are the same as the previous scheme: SMTP and POP / IMAP.

8.2 Simple Mail Transfer Protocol (SMTP)

- *Simple Mail Transfer Protocol (SMTP)*: protocol used to send in the mail push. Uses port 25 / TCP to talk to the MTA and port 587 / TCP to talk to the MSA.
- *Post Office Protocol (POP)*: pull protocol used to read mail, uses port 110 / TCP.
- *Internet Message Access Protocol (IMAP)*: Protocol always used to read mail, is based on port 143 / TCP.
- *RFC* -822: format of the email message. A message in RFC-822 format is a plain text message, that presents only a body that is completely intelligible by a human.
- MIME: multimedia extension of RFC-822, to allow the transmission of data are not only pure text.

8.3 Messages RFC-822

In an RFC-822 has some constraints:

- You can only use the characters of the American 7-bit (US-ASCII 7-bit); the eighth bit is considered not significant, although it is present.
- All lines must be terminated with <CR > <LF >. This is because the various OS use a different line terminator: Windows uses <CR > <LF >, Unix uses <LF >, MAC uses <CR >.
- The messages are composed of a header and a body
  - Header: contains the keywords at the start line, and if a line is longer than a certain size to continue on the next line; continuation lines begin with a space.
  - Body: separated from the header by a line completely empty, and contains the actual message.

8.3.1 Header RFC-822

Note the difference between these two headers:

- From: logical sender of the message, can be freely defined.
- Sender: is the real sender of the message, the' address from which the message was actually sent.

In a message, in the section header, a received line is added for each MTA that is crossed; is a mechanism similar to traceroute. This way you can see the MTA chain.

- **From:** sender (logical)
  Sender: sender (operational)
- **Organization:** organization of the sender
- **To:** destination
- **Subject:** subject
- **Date:** date and hour of sending
- **Received:** intermediate steps
- **Message-Id:** sending ID
- **CC:** copy to
  Bcc: copy (hidden) to
- **Return-Receipt-To:** return receipt to

8.3.2 An example SMTP / RFC-822

```
telnet duke.colorado.edu 25
   Trying .....
   Connected to duke.colorado.edu
   Escape character is '^]'
   220 duke.colorado.edu ...
HELO leonardo.polito.it
   250 Hello leonardo.polito.it ... Nice to meet you!
MAIL FROM: cat
   250 cat ... Sender ok
RCPT TO: franz
   250 franz ... Recipient ok
DATA
   354 Enter mail, end with "." on a line by itself
   From: cat@athena.polito.it (Antonio Lioy)
   To: franz@duke.colorado.edu
   Subject: vacation

   Hello Francesco,
   I renew my invitation to come to my place
   during your vacation in Italy. Let me know
   when you arrive.

   Antonio
   .
      250 Ok
   QUIT
      221 duke.colorado.edu closing connection
      connection closed by foreign host
```

1. This example uses the command telnet to connect to the mail server duke.colorado.edu through port *25*. The server responds with a positive code.

2. The user is presented to the server via the command `HELO leonardo.polito.it`, to which the server responds again with a positive message.

3. `MAIL FROM: cat` is used to define the message sender. Again you receive a positive response from the server because it can not know the real identity of the user.

4. `RCPT TO: franz` defines instead the message recipient. The server takes a little'to respond to this command because it has to check if the specified recipient actually exists within all mailboxes; in any case in this example, the server responds positively.

5. `DATE` is to indicate that the user is about to enter the actual message. The server responds with the code 354, which means that if the message will be formatted correctly will be sent. Once the text has been added to the user inserts a line containing only "."

6. The server checks the message, and because the syntax is correct answers with a positive message.

7. `QUIT` ultimately serves to disconnect from the server.

As you can easily understand from this is an exceptional system in order to send emails false because you have absolutely no control to see the identity of the person who sent this email.

It is possible to enter any address as the sender.

8.3.3 Problems

Securing email is not trivial:

- connectionless system, ie you do not have a direct link between the sender and the recipient. Indeed it comes to system *store-and-forward*, ie the message before it is sent from one node to another is first saved locally and only later is submitted.
- Use of MTA on unreliable which we go to store (even temporarily) the messages.
- level of security of the MS on which is going to store messages before they are read.
- It could implement an encryption system, but would be some problems in the If we were to use a mailing list (How to implement encryption for all members of the mailing-list?).
- Because email is now a system wide use, make changes is not a simple thing to do .

8.4 Mail spamming

Also called Unsolicited Bulk Email (UBE) or Unsolicited Commercial Email (UCE), the term *spam* is used to indicate the e-mail messages side mandates for commercial purposes, but also for purposes of attack (from which comes the need to consider this topic in the field of security). In particular spam emails may contain malware, can be used to operate a phishing attack, etc.

The main problem with spam is that today represents about 88 % of all email traffic. As a result of all this mass of data due to a heavy load on the server (committing CPU and memory) and network (occupying bandwidth unnecessarily), and of course to generate a big annoyance for users.

The word "spam" comes from a Monty Python sketch (British comedy group), in which there was pork in box of very low quality brand spam. From this analogy one can understand why the good mail is instead called "ham".

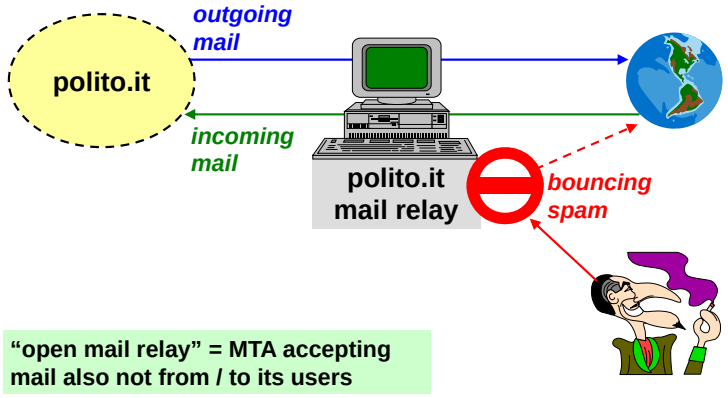
8.4.1 Strategies spammers

Spammers use various strategies to ensure that their messages are accepted:

- Hiding the true sender, however, going to use an existing sender. For example it is common to put the same as the sender e-mail address.
- Send spam through the MTA special
  - Open Mail Relay, who agree to send mail for users who are not part of their domain.
  - Zombie or botnet, instructed to not send the e-mail address of the actual user, but the emails on behalf of spammers.
  - With IP address variable or non-existent, so as to make it difficult to identify the MTA used by spammers.

- Given that one of the strategies anti-spam is to go to look at the text of the e-mail in order to check whether or not containing advertising, trying to make it difficult to identify the message by *content* obfuscation , or try to make the message easy to read by programs (but easily understandable by humans)
  - Error resolved (eg. Viagra = Vi @ gr @)
  - Insert an image instead of text, based on the fact that the automatic recognition programs work well on the text but very bad on the pictures.
  - As one of the anti-spam strategies is to go and see if a message from control has some similarity with other spam messages via the' use of Bayesian estimation techniques, the spammer run the Bayesian poisoning, that fits the body of his email followed by the random text (eg. sentences extracted from a book). This only serves to alter the statistics of words.
  - Enter the text of the spam emails in a (fake) email error, using the fact that not all anti-spam control error messages.

(Open) mail relay



The mail relay is typically the system that controls the inbound and outbound to a particular domain.

Logically it corresponds MTA outbound or inbound:

- Output: must allow the mail generated within the domain of internet reach.
- Login: if someone from the internet decides to send mail to that domain The mail relay must accept it and transfer it to the domain in question.

What a mail relay should not be able to do is to accept the mail from the outside, where the sender wants to be resubmitted to the outside. Since the MTA systems are store-and-forward this operation is theoretically entirely lawful.

The difference between a mail relay normal and an open mail relay lies in the fact that the former is configured so that it can not be used to send mail to users not owned by the managed domain, while the second is misconfigured and may be exploited by spammers to send email on behalf of their users outside the domain.

8.4.2 Strategies anti-spam for MSA

A first strategy is to try to prevent spammers inject the mail in the circuit, going to define the rules that relate to the MSA:

- Never configure their MSA / MTA as an open relay, but restrict their use only to its users.

- Linked to the previous strategy arises the problem of how to identify users. In particular authentication could be done in different ways:
  - IP address of the MUA, but in this case we would have the problem of mobile nodes, the existence of IP spoofing and the possibility of having a malware installed on a valid node.
  - value of the From field, but this is easy to circumvent by means of fake emails.
  - Authentication MUA SMTP level.

8.4.3 Strategies anti-spam for incoming MTA

A system of anti-spam can also be put on incoming MTA, ie the one that controls incoming mail for a particular domain. In particular, the incoming MTA will have to decide whether to reject or accept mail from an external MTA going to check a blacklist or whitelist. This can be done through various solutions.

DNS-based blacklists (DNSBL)

DNSBL The scheme has been standardized in RFC-5782 "DNS blacklists and whitelists." Its operation is as follows:

- You receive a connection request from an external MTA, whose address ABCD
- The question that must be asked is whether this address is known to send spam. This is verified by performing a lookup: nslookup -q = A DCBAdnsbl.antispam.net (there are different DNS domains that provide a service of anti-spam: MAPS RBL, SORBS, etc.). If you receive an answer:
  - NXDOMAIN (No such domain), then the MTA is not a spammer.
  - An address, then the MTA will be a spammer. In particular addresses that are provided are all the type 127.0.0.x, where X is a code that indicates why that node was identified to be a spammer.
    Also, if you run a query lookup type TXT, receiving more information.

Note that once it is placed on a list DNSBL is not trivial to get out: it is necessary to configure your MTA good from the start.

In this regard, the RFC-2142 introduced the obligation to create for each domain a mailbox domain abuse, to which people can send external emails to report abuse in the use of the mail service mail from that domain. This is convenient for the operator of the domain, as it allows him to perform some checks and to take any decision before the domain is placed inside a blacklist.

URI DNSBL

As the above diagram is based on the fact that a certain address is known to send spam, it's not good if the spammer continuously change the sender address of his messages. For this reason, another technique that is often used is to go to accept anyway all email, but before sending them in the mailbox of the recipient running a check on their body going to see if they contain the URI. This scheme is called *URI DNSBL*, and is based on going to establish the reputation of the URI contained within the body of the email.

This scheme is often implemented using the honeypot, which in this case are called spamtrap. These are nothing but the domains that accept email of any kind only to classify them according to any URI containing.

Greylisting

Greylisting is another method to protect users from spam email. A mail server that uses this technique to temporarily reject all e-mail from a sender who does not know. If the email is legitimate, the sender's mail server will retry sending, and this time the operation will be accepted. If the email is sent by a spammer, you probably do not have one more attempt because the mail server, the spammer will have access to thousands (or millions) of email addresses, would pass over not dealing with errors.

This technique has the big disadvantage of delaying not only spam emails, but also the ham: any e-mail you receive will be delayed for a delta (usually equal to 5 minutes). In addition, the incoming MTA also has a higher load than normal because it must keep a table of all the MTA who have contacted in the last few minutes, so you know which ones have already tried sending another.

Nolisting (Poor man's greylisting

Due to an increased workload required to servers, has also developed the technique of *nolisting*. It is always based on the theory that spammers have no time to lose, and then do not go to try all the MX (Mail eXchanger, ie un'incoming MTA in the terminology of the DNS) defined for each domain. In particular domains using this technique go to define different MX:

- 1° MX: never answers.
- 2° MX: operating normally.
- 3° MX: never answers.

This is done because spammers typically use the first or the third MX, do not try them all.

This technique has the advantage of blocking, or delay, spammers but also causes a delay dell'ham. It, however, has the advantage of not overloading further MTA because you will only receive mail valid and do not need to keep track of all the external MTA that have contacted him.

DomainKeys Identified Mail (DKIM)

Technical anti-spam defined in several RFC, is based on using a whitelist, or a list containing a list of MTA officially delegated to send mail for a certain domain.

The outgoing MTA of a specific mail domain must ensure cryptographically

- The identity of the sender.
- The integrity (partial) of the message. This is done to prevent a valid message can be modified by an intermediate MTA, turning it into a spam message.

These characteristics are ensured by the use of a digital signature. This signature:

- is affixed dall'ougoing MTA or directly the MSA.
- It covers some of the header RFC-822 and a part of the body.
- must be verifiable by the use of a public key . Usually what is done is to put the public keys in the DNS.

This technique is used in an ever-increasing: for example, is used by Gmail and Yahoo to identify mail that has been sent by one of their MTA.

Allows you to discard messages with a fake sender, and then performs a function of anti-spam and anti-phishing in part, in the sense that you can not send messages on behalf of another person.

Sender Policy Framework (SPF)

strategy of anti-spam defined in RFC-4408, is based in a sense always on cryptography.

The incoming MTA have already declared in the DNS through the MX record, but you do not have any information on what the outgoing MTA for that domain. In this regard, we have added a special record in DNS to declare what are the outgoing MTA valid.

```
$ nslookup -q=txt polito.it.
polito.it text = "v=spf1 ptr ~all"
$ nslookup -q=txt gmail.com.
gmail.com text = "v=spf1 redirect=_spf.google.com"
$ nslookup -q=txt _spf.google.com.
_spf.google.com text = "v=spf1 ip4:216.239.32.0/19
   ip4:64.233.160.0/19 ip4:66.249.80.0/20 ip4:72.14.192.0/18
   ip4:209.85.128.0/17 ip4:66.102.0.0/20 ip4:74.125.0.0/16
   ip4:64.18.0.0/20 ip4:207.126.144.0/20 ip4:173.194.0.0/16
   ?all"
```

- 1° example: any computer within the Poles who have a record ptr valid, can send outgoing e-mail.
- 2° example: list of all of the addresses IPv4 of outgoing MTA valid for Google.

8.5 Extended SMTP (ESMTP)

Another strategy used to block spam is to prevent spam is injected into the system; this must be done by authenticating sull'MSA the MUA wishing to send mail. However in SMTP base there are strategies of this kind.

In this regard it was decided to extend the protocol, leading to the birth of the ESMTP protocol. This protocol does not change the basic protocol and even the transmission mode on the channel. What is changed is the greeting with HELLO
centerline *EHLO hostname*

If the receiving server speaks too ESMTP, then it will have to declare that supports extensions, one per line, in its response; otherwise respond saying unknown message.

8.5.1 Examples ESMTP positive

Mailer without ESMTP extensions

```
   220 mail.polito.it - SMTP service ready
EHLO mailer.x.com
   250 Hello mailer.x.com - nice to meet you!
```

1. You run a connection request to the server of the Polytechnic.
2. It says it will use the extended version of the protocol by sending the message EHLO.
3. The server responds with a positive message (250), which is indicating that he is able to speak ESMTP. However, this message does not contain any extension.

Mailer with ESMTP extensions

```
   220 mail.polito.it - SMTP service ready
EHLO mailer.x.com
   250-Hello mailer.x.com - nice to meet you!
   250-SIZE 26214400
   250 8BITMIME
```

1. You run a connection request to the server of the Polytechnic.
2. It says it will use the extended version of the protocol by sending the message EHLO.
3. The server responds with a positive message; 250- the code is to indicate that the current row will be followed by another. Particularly in the next two lines are set extensions supported:
   - *SIZE*: Extension important because it states what is the maximum length of an e-mail that the server will accept.
   - 8BITMIME: extension that lets you know that the receiving mail server is not going to change the eighth bit; in high words you can send data that are meant also the eighth bit.

8.5.2 Example ESMTP negative

```
   220 mail.polito.it - SMTP service ready
EHLO mailer.x.com
   500 Command not recognized: EHLO
```

1. You run a connection request to the server of the Polytechnic.
2. It says it will use the extended version of the protocol by sending the message EHLO.
3. The server responds with a negative message (500), which means that it is able to speak only the basic version of the protocol.

8.5.3 SMTP-Auth

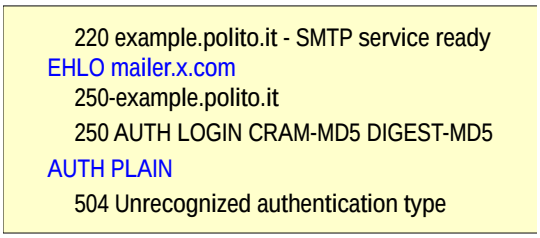
Among the various extensions that have been defined for ESMTP, there is the extension SMTP-Auth (RFC-4954) used to do authentication.

This extension introduces the AUTH command over some options MAIL FROM command.

In general, however, the aim is to authenticate a client (MUA) before accepting messages. This is very useful as anti-spam strategy:

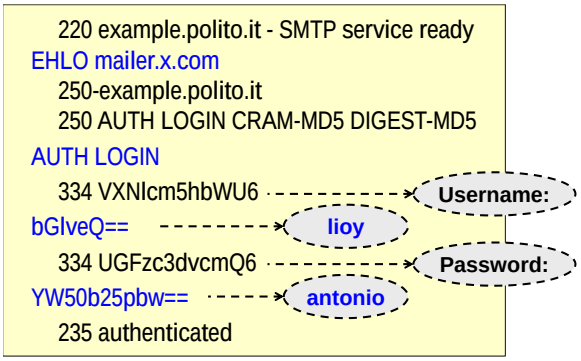
- After the EHLO command the server sends the authentication mechanisms supported.
- The client chooses one.
- runs the authentication protocol.
- If authentication fails, the channel is closed. If instead the authentication is successful you are assured that it is a valid MUA who is entitled to use the MTA to send mail.

Example AUTH negative



1. You run a connection request to the server of the Polytechnic.
2. It says it will use the extended version of the protocol by sending the message EHLO.
3. The server responds with a positive message (250), and in particular claims to support authentication using the methods LOGIN, CRAM-MD5 and DIGEST-MD5.
4. The client should choose between one of the authentication methods offered by the server, but not by supporting no one tries to authenticate using the method PLAIN.
5. The server will respond with an error message, and then authentication will fail.

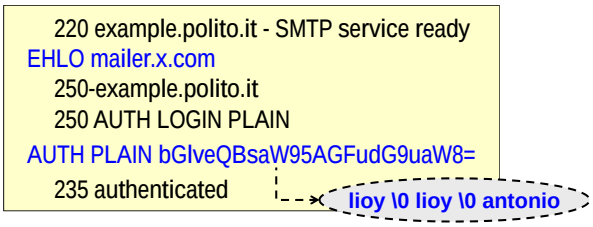
AUTH: Method LOGIN



1. You run a connection request to the server of the Polytechnic.
2. It says it will use the extended version of the protocol by sending the message EHLO.
3. The server responds with a positive message (250), and in particular claims to support authentication using the methods LOGIN, CRAM-MD5 and DIGEST-MD5.
4. The client chooses to authenticate using the method LOGIN.
5. At first glance it may seem that the following lines are not intelligible. But you have to remember that e-mail using a standard 7-bit channel, and therefore what has been done is to map all data transferred on 7 bits; especially you go to use BASE64 encoding.
6. The server checking that the data entered by the user are correct, it authenticates.

This method is not particularly safe, because anyone sniffing the network must semplicmente perform decoding base-64 data to obtain a username and password of the user.

AUTH: Method PLAIN



LOGIN variant of the method that avoids making four message exchanges because username and password are sent directly when you run the command AUTH PLAIN, everything always encoded in BASE64.

$$AUTH\ PLAIN\ id\ \_pwd\ _{B}ASE64$$

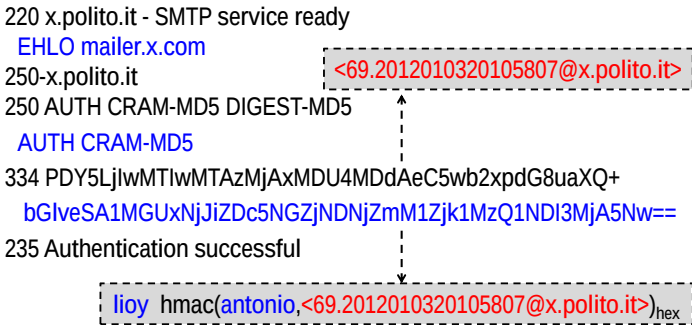
The string id _pwd is defined as
centerline *[authorize _id] \0 authentication _id \0 pwd*

- *[authorize _id]*: identification authorization (Optional).
- *authentication _id*: Identified client authentication.
- pwd: password client.

Aside from making the authentication procedure faster, this method does not introduce any additional level of security than the method LOGIN. This means that LOGIN and MAIN would be advised unless they are used over secure channels.

AUTH: Method CRAM-MD5

Challenge-Response Authentication Mechanism (CRAM) is an authentication method based on the algorithm HMAC-MD5.



The protocol is essentially composed of three phases:

- *Challenge*: CRAM-MD5 in authentication, the server sends a challenge string to the client.
- Response: The client responds with a string with this structure:
  - The string sent by the server, encrypted with BASE64, is decoded.
  - The decoded string is encrypted with HMAC-MD5 using the user's password as the secret key.
  - The challenge coded is converted to hexadecimal digits.
  - The username and a blank space is added at the beginning of the string hex
  - The concatenation is BASE64 encoded and sent to the server

- Comparison: the server, knowing the password associated with the user, running the same method of the client and if the string is calculated the same as that sent by the client then authentication is granted.

8.5.4 Securing SMTP with TLS

If you want to use the methods and PLAIN LOGIN CRAM-MD5 instead must operate over a secure channel. But even if we were to use CRAM-MD5 should use a secure channel, because this method solves the problem of authentication but does not solve the problem that someone can sniff the message in transit, going to read it and maybe even alter it .

In this regard it was decided to apply to the TLS protocol SMTP (RFC-2487). In particular, we have added a new command called STARTTLS, which is also an option of EHLO.

If the negotiation is successful resets the status of the protocol; in other words it starts from the message of EHLO, and supported extensions may be different.

In case instead the security level negotiation was deemed insufficient:

- If the client to feel it, then it will send the QUIT command and exit.
- If the server is instead to consider it, then it will respond to every command received from the client with the error 554 (two refused to low security). It thus leaves to the client always the burden of closing the channel.

```
    220 example.polito.it - SMTP service ready
EHLO mailer.x.com
    250-example.polito.it
    250-8BITMIME
    250-STARTTLS
    250 DSN
STARTTLS
    220 Go ahead
... TLS negotiation is started between client and server
```

Note that the TLS channel could be negotiated between a MUA and MSA, but also between two MTAs. This means that this solution does not provide an end-to-end but hop-by-hop. As a result you could have a secure channel to a section, and not on the next leg. In other words the use of this extension does not provide any guarantee that the entire route taken by an e-mail message is protected.

For this reason, this extension is not used very much. It is typically used to create a secure channel between a MUA and MSA in case you wanted to use the method LOGIN or PLAIN, or in case you want to try to avoid sniffing at least within the local network.

8.6 Security Services for e-mail messages

Since the use of TLS does not guarantee an end-to-end, it is necessary to use other security mechanisms that protect the inherently message is the message to be end-to- end, its transportation is done hop-by-hop.

The services needed to make sure the e-mail really are:

- Integrity, even if you have no direct communication between sender and receiver. In this way the message can not be changed or during transmission or on intermediate servers.
- *Authentication*, to be able to positively identify the sender and thus avoid the fake email.

- *not* repudiation, in so that the sender can not deny having sent a message.
- *Confidentiality* (optional) so that messages are not legible both in transit and in the recipient's mailbox.

8.6.1 Security email - Ideas guide

- *No changes to the current MTA*: it is essential to encrypt messages in order to avoid problems in crossing the gateway.
- *No changes to the current UA*: ability to have user interfaces uncomfortable.
- *Edit the current UA*: better user interface as will complement any function.

The major MUA exist today (Outlook, Thunderbolt, etc.) Have already directly implement all the functions that are used to make secure e-mail, and therefore the user interface is very simple to use.

From a technical standpoint, the tools used to make security of email are:

- Symmetric algorithms, used to make confidentiality. This will make it possible to encrypt the message (even if you have very large) with a key message.
- asymmetric algorithms, used to encrypt and exchange with the recipient the symmetric key choice, but also for the portion of the digital signature.
- public key certificates (eg. X.509), to support the function of non-repudiation.

In this way you will have the security of the message will be based only on the safety of the AU, and not on that of the MTA (unreliable).

8.6.2 Types of secure messages

There are four different types of messages:

- Clear-signed: plaintext message (and therefore readable) + digital signature (which can be placed as attached or can also be an integral part of the message). The signature will be verified only by those who have a MUA safe, ie a MUA that supports the security protocol used.
  This is the e-mail message that there is generally more secure because the message is readable by anyone (you do not need to know whether the recipient supports secure mail or less), and is also a digital signature that guarantees integrity, authentication and non-repudiation.
- *Signed*: plaintext + signature, both encoded using base64 or such other methods . To verify the signature, but also just to read a message, you need to decode the message; therefore it is necessary to have a safe MUA, or need to make an effort manual (you save the message as a text and then it is decoding with a suitable program).
- *Encrypted / enveloped*: ciphertext + encrypted keys (with the recipient's public key), all coded (typically base64). Only those who have a MUA safe and has the cryptographic keys needed to decrypt the message will go.
- *Signed and enveloped*: message + encrypted keys, all encoded.

Posts sure - Create

- canonicalization: do not closely linked to security, is to put the message in a standard format, which is independent of the OS, from the particular host, network etc.
- *Message Integrity Code (MIC)*: the addition of this code ensures integrity, authenticity and non-repudiation. Typically this code is created by concatenating the message with a hash of the message encrypted with the sender's private key.

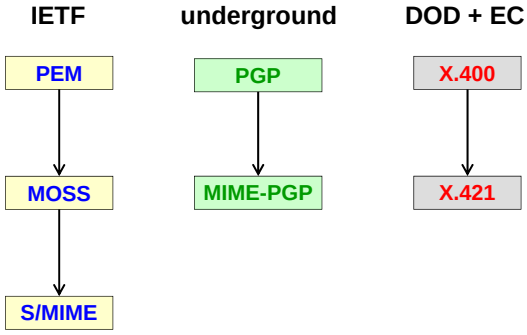
$$msg + c(K_{pri\_sender}, h(msg))$$

- Encryption: this is done in case you wanted to also have confidentiality. Typically encryption is done as follows:

$$c(K_m, msg) + c(K_{pri\_sender}, K_m)$$

  - K_m = message key generated on the fly.
  - $K_{\text{pri\_sender}}$ public key receiver.
- *Code*: operation performed in order to avoid alterations by the MTA intermediate. Typically base64 encoding is used (in the past it was also used as uuencode and binhex).

8.7 Formats secure email



Initially, the IETF has gone to define the format PEM, which served to make safety on mail messages RFC-822 base.

Subsequently, following the appearance of MIME, which allowed to have attachments, was developed MOSS, which is nothing but a PEM applicable to the attachments. MOSS however, has not been very successful because from a commercial point of view the RSA has proposed an alternative format, similar in concept but different in implementation, called *S / MIME*. This is now the de facto standard for email security.

The underground world, however, has not accepted these standards, mainly for the fact that they require the use of X.509 certificates. It'was then developed an alternative format called PGP, followed by PGP-MIME to support MIME.

The military (Department of Defense, DOD) and partly also the European institutions have long supported X.400, and later version for X.421 support to multimedia introduced by MIME.

8.7.1 Pretty Good Privacy (PGP)

PGP is a system to do authentication, integrity and confidentiality of generic data. Originally was applied to e-mail messages, but also to private files. Its objectives were the same as those of PEM, and the structure was very similar but was much craftsmanship.

This was due to the fact that PGP was developed by a single person, Phil Zimmerman, in order to protect the exchange of messages between friends and upload files. PGP was developed at a time when the Internet was very different from today. To exchange files resorted to so-called

BBS (Bulletin Board System), ie the server to which you are connected via modem, and when people put the files that were in effect the message. These messages could be public or private, that is intended only to some users of the BBS. The problem was that the managers of the BBS could read the messages and edit them. In this scenario Zimmermann has developed a system of protection which provided a minimum integrity and authentication, so as to avoid changes by the operators, and possibly confidentiality, so that the messages were readable only by certain users. This protection system could also be applied to e-mail as conceptually systems were very similar. PGP was originally defined in RFC-1991, which contains information on the scheme chosen by Zimmermann. More recently PGP has been defined in RFC-4880, with the definition of the OpenPGP standard.

One of the features of PGP is to not use X.509 certificates, but to use a certification scheme of the keys very peculiar when it comes to friends (trusted and untrusted), and to use an algebra propagation of trust. PGP is famous because it is the most widely used system in the world, developed for all platforms. PGP and Zimmermann have become long been a symbol of freedom on the Internet against the attempt of the government to control the Internet itself.

Phil Zimmermann

Phil Zimmermann released PGP as freeware in 1991. The trouble began when a copy of the program has been loaded on a server in Finland. Because in those years there was the ban on the export of cryptographic material, the US government has jailed Zimmermann.

Zimmermann was immediately declared innocent, and the lawsuit has been going on for some time (Zimmermann has received support from many people, who were also donations so that he could pay his lawyers) until in 1996 the lawyer of the US Government has waived his right to persecute him. Following this incident, and proud to have won his case, Zimmermann decided to found the PGP Inc., which will then be acquired by NAI.

Following the acquisition PGP has undergone a number of changes; Zimmerman continued to be in a sense the guarantor that the system continues to be open, free for those who wanted to use for personal and protected. Since the code was open source PGP, however, been found that had been added in the backdoor code, which allowed to decrypt messages without your consent. After this has been created a major controversy on the internet, and the statements of Zimmermann on being totally unconnected with the facts have not been well received because he being the Chief Scientist authorizing all changes some responsibility had definitely . This caused the total loss of confidence in it.

Zimmermann then decided to found a new company in August 2002, the PGP Co., which produces PGP but only for Windows and also you need to pay a license to use it.

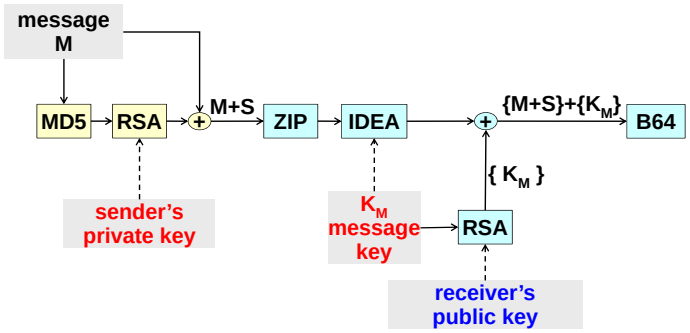
PGP - Algorithms (up to v. 2.6)

Until v. 2.6 (original version of Zimmermann) PGP has a very simple structure, with the use of fixed algorithms:

- IDEA for symmetric encryption.
- for the MD5 digest.
- RSA for the asymmetrical part, both for the digital signature is for the exchange of the symmetric key.

These implementation choices were made because all these algorithms are free for non-commercial use.

Example PGP 2.6 - Signature Encryption +



1. Calculated MD5 digest of the message, which was then encrypted RSA with the sender's private key.
2. Concatenation of the signed message calculated above.
3. ZIP data to reduce its size and eliminate redundancies.
4. Data encrypted with IDEA using a message key generated on the fly.
5. key message encrypted with RSA public key of the recipient.
6. Concatenation data genrati in points 4 and 5, and subsequent base64 encoding.

PGP - Certification of keys

The certificate is nothing but the public key of the person you are certifying, signed by all the people who trust this person. So there is not a single signature made by the CA, but there are so many signatures made by all the "friends" of this person.

The trust is propagated transitively with a certain degree of approximation. In other words, means that you make a classification of friends according to a number of categories:

- Completely: friends you trust completely.
- Partially: friends we trusts partially.
- Untrusted: friends whom you do not trust.
- *Unknown*: rest of the world.

The pattern in the example 8.2 is called web of trust. This scheme is similar to the way we trust the social and human relations. Not so trivial revoke the trust at a time when a private key is compromised.

In the example we have that C is a friend of B, of which he trusts completely. Because I I trust you completely of B, the transitive property of the trust also I'm going to trust C (although for me it is a stranger).

You must have at least 3 partial confidences so that confidence spreads. The user has the key signed by F, G and H, and for this reason I will mark you as a user partially trusted.

M has only one member in common, F, with which it has a partial trust. Consequently M does not exceed the threshold. FloatBarrier

PGP - Key distribution

The keys are kept personally by each user in the so-called key-ring. They are distributed by the owners, such as in the famous PGP party, or going to load them in a key-server (which distributed them via http, smtp, finger, ...).

There were also some projects to distribute keys using X500 or DNS. For example there is the network pgp.net, on which you can go to download the keys of users.

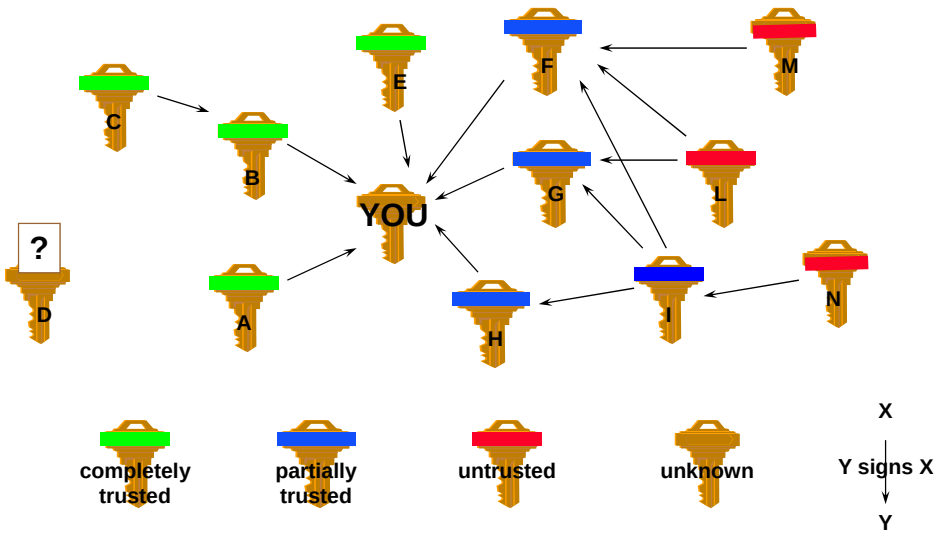


Figure 8.2: Web of trust.

PGP & NAI

PGP was acquired from NAI in December 1997.

First problem that NAI has had is that the algorithms chosen by Zimmermann were free for personal use, but a fee for commercial use. Therefore it was necessary to replace the algorithms with others who were also free for commercial use; in particular, it was decided to base the signature of DSA, DH key exchange and encryption of 3DES.

PGP has been integrated in many e-mail systems, and has also had some attempt to introduce in the corporate market. But the companies did not like the idea of PGP party, and for this reason NAI has suggested the creation of pseudo-CA in the sense of having a super-signer (a place to make your key signed by all the friends you did sign by the head of the security company). But this has not been successful, and in September 1998 NAI has decided to develop a version that supported directly X.509 certificates; However, this has never been done.

In August 2002 NAI sold all rights to dul program PGP PGP Co.

Gnu Privacy Guard (GPG)

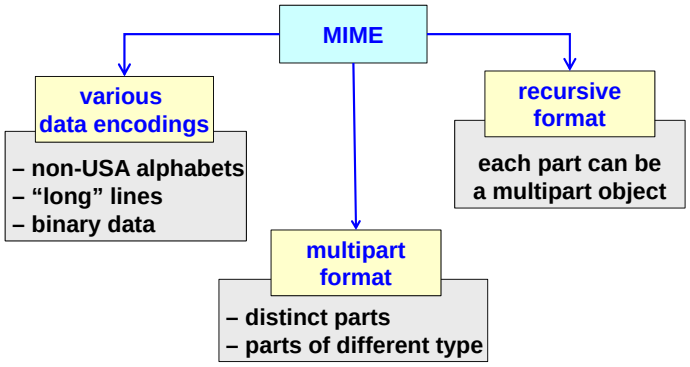
Because PGP Co. has decided to no longer release PGP freeware version, and especially to develop it only for the Windows platform, the Internet community has felt betrayed and has developed an alternative solution called GPG.

GPG is a complete rewrite of PGP under GPL and free of any patented algorithm. It interoperates with both messages PGP 2.x with OpenPGP messages.

It has the support of many cryptographic systems, many front-end graphics and is available for virtually any platform.

Note that PGP and GPG has remained niche solutions, used by a few people, but it seems that something is changing. In August of 2014, Gmail and Yahoo, as a result of the scandal NSA, said that they will integrate into their system of web-mail OpenPGP; at this time this solution is under development. This is definitely a positive thing, because it will have secure email for users web-mail, but will no doubt create interoperability issues (S / MIME uses a completely different format and is based on X.509 certificates). It is therefore not clear whether you will have two different systems (S / MIME and PGP for corporate systems for users) or you will find solutions to ensure interoperability.

8.7.2 Multipurpose Internet Mail Extensions (MIME)



MIME is used to extend the support that typically were only textual formats so that they can also support other types of information. In particular MIME was born as a supplement of the mail because the mail was initially purely textual.

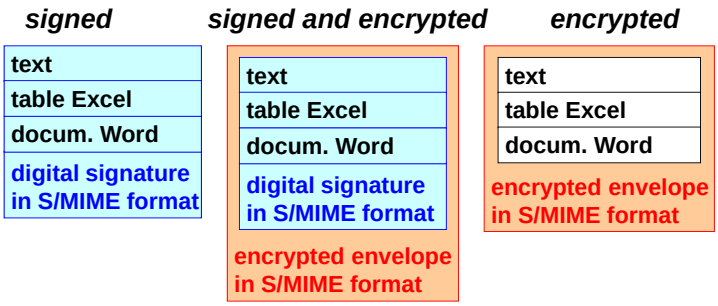
MIME provides essentially three major innovations:

- *Various encodings of data*
  - Ability to use scripts other than the US 7-bit. This is very important because for example if a message you want to have accented letters must also use the eighth bit, that RFC-822 was instead used for other purposes.
  - Possibility to have long lines, while basic you could not have lines longer than 72 characters.
  - can be inserted into a message generic binary data, such as images, videos, etc.
- *Introduction of the concept of multipart*: an object such as the e-mail message may contain within it several distinct parts, each with a different type. This concept is used for example when sending a text message with the so-called attachments: actually are not attached, are all separate parts, all composed within a single message.
- *recursive* Format: each of parts of an object can themselves be objects MIME.

All these concepts are in general very important for e-mail, but also as regards safety. First, the possibility of adding binary data is very important because it allows for example to encode in a message its digital signature. FURTHERMORE typically the digital signature is something added to the message; so then have a multipart format allows you to add a digital signature as part of the message. Finally recursion can be useful in the case of encryption, meaning that the original MIME message may be contained in another object MIME encrypted.

8.7.3 Mailbox multimedia safe - MOSS or S / MIME

MIME has been widely used to make email security. In particular they were created two formats, MOSS and S / MIME, offering digital signature capabilities and encryption MIME messages through the use of X.509v3 certificates.



In the image to the left you have a generic message format MOSS or S / MIME parts containing MIME and last part corresponding to the digital signature (hash calculated over all parts MIME encrypted with the sender's private key).

In the image to the right instead you have an example of a coded message, in which the original message MIME was inserted inside another MIME message after having done the encryption; in other words, the encrypted message has been added as binary data within another MIME message.

Finally in the middle image you have an example of a message signed and encrypted, created by applying the two previous techniques.

8.7.4 RFC-1847 (MOSS)

With the RFC-1847 MOSS has defined several extensions to the safety of MIME messages. In particular, it had gone to define both the part of the digital signature is the part encryption. However the part of encryption is no longer used, while for the digital signature continues to be used in part with the same pattern.

The digital signature is made going to add to the message MIME content type *multipart / signed*, that states that the message is made up of different parts, of which the last is the digital signature.

```
Content-Type: multipart / signed;
    protocol = "TYPE / STYPE";
       textsl micalg = "...";
          boundary = "..."
```

You also have the statement of the manner in which you go to generate the digital signature (with an indication of a type and subtype) and MIC algorithm, ie the hash algorithm used to compute the digest of all the previous parts the digital signature. Finally you have the declaration of the boundary, that is where it ends and begins the next part.

They made two body part:

- The one to be protected;
- The signing, which content type as the type and subtype declared earlier.

This scheme should be generally very good, but there could be problems if a gateway alters messages. This is the pattern still used when you want to have the message type clear signed, which is readable to all but those that contain digital signature.

8.7.5 S / MIME

S / MIME is the main standard used today for the safety of MIME messages.

For a certain period of time and the MOS S / MIME have been competing in the sense that MOS was promoted by IETF while S / MIME was promoted by RSA (going to use a number of proprietary solutions). The v2 of S / MIME was published only as a series of informational RFC, so without any endorsement by the IETF. With the passage of time has increasingly been the trend to move to S / MIME, which from v3 has become an IETF proposed standard.

RFC-2634

In addition to the services based on S / MIME, in' RFC-2634 are also indicated services to offer more security. In particular, there are four areas where you have had extensions:

- *Signature* for receipt of a document, so that the recipient of a message can not deny having received it.
- Security label textit , the mechanism through which it goes to classify each message according to a certain hierarchy of security. For example in the military messages are classified in unclassified (readable by anyone), classified (reserved only to those who have a certain authorization), secret and top-secret.
  This mechanism is generally used to send messages using only of lines and MTA that have at least the same level of security. The security label is only a suggestion, however, because it is said that the infrastructure has the capacity to base the routing based on this feature.
- *safe* Mailing-list, which solves the problem of how do to encrypt messages destined to a mailing-list (where you are not available to the public keys of the recipients). This, however, introduces a weakness because the message is not encrypted end-to-end but the manager of the mailing list could potentially go to read the contents.
- *Signature of the certificate attributes.*

Architecture of S / MIME

From an architectural point of view, S / MIME is based on:

- *PKCS-7* (S / MIME v2) and *CMS* (S / MIME v3): specifies the cryptographic features and types of messages (equivalent to PEM).
- *PKCS-10* in case you wanted to use the same e-mail to make the request for a certificate.
- X.509 Certificates in support of signatures and encryption.

S / MIME - Algorithms

The algorithms used are:

- Digest: SHA-1 (mandatory) or MD5.
- Digital Signature: DSS (required) or digest + RSA (solution used) .
- exchange keys: Diffie Hellman (required). In fact, the common practice is to calculate a key message cifrandola with the RSA key of the recipient.
- Encrypt Message: 3DES or RC2 / 40 (both required).

The preference for DSS and Diffie-Helmann, algorithms generally little used, comes from the fact that S / MIME has been standardized by the IETF, which wished to specify as mandatory algorithms that were free from patents, which did not require the payment of royalties implement them.

MIME Type

To support S / MIME have defined the new MIME types. Specifically:

- *application / pkcs7-mime*, used to create
  - encrypted messages (the message is encoded as a envelopedData PKCS-7).
  - Messages signed binary (PCKS signedData-7), intended only for users S / MIME because the message is inside the envelope PKCS-7 (not clear).

– Messages that contain only a public key, or the so-called PKCS-7 signedData "degenerate" presenting the data section nothing. The purpose is only to provide the recipient certificate.

The standard extension to be used in combination with such .p7m is, and has always encoded in base64 because otherwise it could be transmitted on a normal SMTP channel.

- *multipart / signed*, used to create messages signed also intended users not S / MIME. With this solution, the message is clear, and the last part is the signature MIME (RFC-18-47). The signature has standard extension .p7s, and is encoded in base64.
- *application / PKCS10*, you need to send certificate requests to a CA. Also this is a base64 encoded.

S / MIME - Sample signature

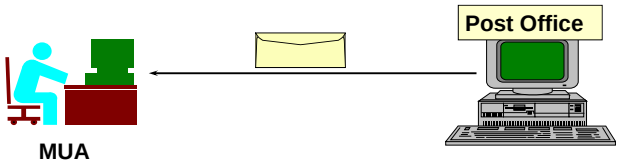
```
Content-Type: multipart/signed;
 protocol="application/pkcs7-signature";
 micalg=sha1;
 boundary="-----aaaaa"

-----aaaaa
Content-Type: text/plain
Content-Transfer-Encoding: 7bit

Ciao!
-----aaaaa
Content-Type: application/pkcs7-signature
Content-Transfer-Encoding: base64

MIIN2QasDDSdwe/625dBxgdhdsf76rHfrJe65a4f
fvVSW2Q1eD+SfDs543Sdwe6+25dBxfdER0eDsrs5
-----aaaaa-
```

Naming in S / MIME

In order to use X.509 certificates in mail support is essential that within the certificate there is the e-mail address. Others identified are not useful, why not associate the key with the email address.

To this end, in the S / MIME v2 were advised to insert a field *Email* = (or = E) as part of the DN. It was also possible to optionally use the extension subjectAltName encoded rfc822; with v3 of S / MIME latter practice has become mandatory.

8.8 access protocols to MS



In client-server mode the MUA has the need to make a query to its Post Office (MS) to determine if there is mail addressed to him.

In a scheme like that you have a sert of problems:

- User authentication by the server in order to avoid giving access to the box to someone who is not the rightful owner.

- Authentication server from the client, so that the latter can be sure you have connected to his real Post Office.
- confidentiality / integrity of e-mail, both on the server is in transit in the network between the Post Office and the MUA .

The protocols used to access the MS are essentially two:

- *Post Office Protocol (POP)*: There are two versions, POP-2 (no longer used) and POP-3. User authentication is done through the use of a password in the clear.
  There is a variant called APOP, in which the user authentication is done through a challenge. Some servers also support the variant KPOP, in which you can not do mutual authentication with the Kerberos tickets.
- *Internet Mail Access Protocol (IMAP)*: the default authentication is given a username and password in the clear. Optionally you can use OTP, Kerberos or its generalization GSS-API.

8.8.1 Example POP-3

```
    telnet pop.polito.it 110
+OK POP3 server ready <7831.84549@pop.polito.it>
    USER lioy
+OK password required for lioy
    PASS antonio
+OK lioy mailbox locked and ready
    STAT
+OK 2 320
.........
    QUIT
+OK POP3 server signing off
```

1. Connection to the Post Office through port 110.
2. The server responds with a + OK (no longer used as the numeric codes in SMTP) and presented to the user.
3. The user enters your username preceded by the keyword *USER*.
4. The server requires the user to enter the password.
5. The user enters the password preceded by the keyword *PASS*.
6. The server does verification user-entered data, and provides the user with access to your mailbox.
7. The user uses the command *STAT* to request the status of the mailbox.
8. The server responds by saying they are two unread messages, for a total of 320 bytes.
9. The user runs the command *QUIT*.

It is easy to see that whoever is able to sniff the communication channel will switch to clear the username and password. POP-3 protocol is therefore a highly insecure.

8.8.2 APOP

The variant APOP introduces a new command, called precisely APOP, which replaces the pair of commands USER + PASS. This command introduces a mechanism to challenge symmetrical, where the challenge is the part of the line hello enclosed in angle brackets (including the brackets). The command syntax is as follows:
Centerline APOP user response-to-challenge

The answer is nothing but a keyed digest, ie *response = MD5 (password challenge +)*. The answer is encoded in hexadecimal, because otherwise it could be transmitted on a channel ASCII.

The most famous was that supported APOP Eudora.

Example APOP

```
    telnet pop.polito.it 110
+OK POP3 server ready <7831.84549@pop.polito.it>
    APOP lioy 36a0b36131b82474300846abd6a041ff
+OK lioy mailbox locked and ready
    STAT
+OK 2 320
.....
    QUIT
+OK POP3 server signing off
```

1. Connection to the Post Office through port 110.
2. The server responds with a + OK and is presented to the user.
3. The user decides to authenticate using the APOP command, and in particular used as challenge the characters <...> previously sent by the server.
4. The server does the same calculation of the user, and gives access to the mailbox.

Although APOP solves the problem of sniffing the username and password, all other data remain in the clear.

8.8.3 IMAP

By default, the IMAP protocol is based on a very weak authentication process:
Centerline *LOGIN user password*

However, there is the option to use one of these 3 commands below:

- *AUTHENTICATE KERBEROS _V4.*
- AUTHENTICATE GSSAPI.
- AUTHENTICATE SKEY.

The only case in which there is mutual authentication when using Kerberos.

8.8.4 RFC-2595 (TLS For POP / IMAP)

Of all the possible options usually either IMAP or POP using the authentication method based on light transmission in the user name and password, and for this reason are subject to many attacks.

It was then decided to use TLS to protect the channel on which then will be subsequently used the POP or IMAP protocol. In particular, the RFC-2595 documents on how to use TLS on these channels.

This RFC has added two new commands, STARTTLS for IMAP and POP3 for STLS, to be executed immediately as the first command in order to transform the normal TCP channel in a secure channel with TLS. Only at this point will be transmitted username and password.

This solution has the benefit not only to protect your username and password, but also to protect all transfers of mail and to also have authentication sever (mandatory TLS).

8.8.5 Separate ports for SSL / TLS?

When you go to configure an email client to connect to a server is normally possible to choose between three different options:

- Use a TCP channel, without any security.
- Use STARTTLS, which allows you to turn the channel created earlier in a secure channel.
- Use SSL, with whom you go to first create the SSL channel on a dedicated port, then this will run the application protocol.

The choice to use TLS or SSL influence on the fact that you will need to separate ports. In fact if you use TLS will be possible to use the same port, but if you use SSL you will need two separate doors, one to access the service normally and the other to access it via SSL.

Separate doors for the delivery of safe / unsafe is normally not recommended by the IETF for the following reasons:

- Imply different URLs (http and https).
- Imply a model safe / unsafe not perfectly correct. For example, if you use https, the resulting channel may not be safe because it could be a channel 40-bit; This implies that although it uses https not you are assured of having a secure channel.
- It is not easy to implement the idea of "use SSL if available", because it would mean making two attempts to connect: If you are unable to connect to the SSL port, using the normal one. This because of rellantamenti in activating the connection.
- You go to double the number of ports required.

In reality have different ports for the delivery of safe / unsafe has some advantages:

- Simplicity filtering firewall packet-filter on, as you just enable / disable ports well determined.
- Perform SSL before the application protocol allows to perform client-authentication, which allows you to not expose applications to attacks.

Appendix A

Definitions

A.1 Introduction to the security of ICT systems

ICT security ✡ the set of products, services, organization rules and individual behaviours that protect the ICT system of a company

availability ✪ the property of being accessible and useable upon demand by an authorized entity

accountability ✯ the property of a system or system resource that ensures that the actions of a system entity may be traced uniquely to that entity, which can then be held responsible for its actions

authentication ☆ the process of verifying a claim that a system entity or system resource has a certain attribute value

peer entity authentication ✪ the corroboration that a peer entity in an association is the one claimed

data origin authentication ✪ the corroboration that the source of data received is as claimed

repudiation ✪ denial by one of the entities involved in a communication of having participated in all or part of the communication

authorization ☆ an approval that is granted to a system entity to access a system resource

access control ☆ protection of system resources against unauthorized access

privacy ✪ the right of individuals to control or influence what information related to them may be collected and stored and by whom and to whom that information may be disclosed

confidentiality ✪ the property that information is not made available or disclosed to unauthorized individuals, entities, or processes

data integrity ✪ the property that data has not been altered or destroyed in an unauthorized manner

replay attack ☆ an attack in which a valid data transmission is maliciously or fraudulently repeated, either by the originator or by a third party who intercepts the data and retransmits it, possibly as part of a masquerade attack

security service ☆ a processing or communication service that is provided by a system to give a specific kind of protection to system resources

asset ✡ the set of goods, data and people needed to offer an IT service

vulnerability ✡ weakness of an asset

threat ✡ intentional or accidental event which can produce the loss of a security property by exploiting a vulnerability

(negative) event ✡ occurrence of a threat of type 'accidental event'

attack ✡ occurrence of a threat of type 'intentional event'

incident ✯ a security event that compromises the integrity, confidentiality, or availability of an information asset

breach ✯ an incident that results in the disclosure or potential exposure of data

data disclosure ✯ a breach for which it was confirmed that data was actually disclosed (not just exposed) to an unauthorized party

hacker ✡ a person who enjoys exploring the details of programmable systems and how to stretch their capabilities, as opposed to most users, who prefer to learn only the minimum necessary

cracker ✡ a person who exploits its own computer knowledge to break security on a system

masquerade ☆ a type of threat action whereby an unauthorized entity gains access to a system or performs a malicious act by illegitimately posing as an authorized entity

spoof ☆ attempt by an unauthorized entity to gain access to a system by posing as an authorized user

wiretapping ☆ an attack that intercepts and accesses information contained in a data flow in a communication system

passive wiretapping ☆ only attempts to observe the data flow and gain knowledge of information contained in it

denial of service ☆ the prevention of authorized access to a system resource or the delaying of system operations and functions

distributed computing ☆ a technique that disperses a single, logically related set of tasks among a group of geographically separate yet cooperating computers

zombie ☆ an Internet host computer that has been surreptitiously penetrated by an intruder that installed malicious daemon software to cause the host to operate as an accomplice in attacking other hosts, particularly in distributed attacks that attempt denial of service through flooding

shadow server ✡ host that manages to show itself (to victims) as a service provider without having the right to do so

active wiretapping ☆ attempts to alter the data or otherwise affect the flow

hijack attack ☆ a form of active wiretapping in which the attacker seizes control of a previously established communication association

man-in-the-middle attack ☆ a form of active wiretapping attack in which the attacker intercepts and selectively modifies communicated data to masquerade as one or more of the entities involved in a communication association

social engineering ☆ euphemism for non-technical or low-technology methods, often involving trickery or fraud, that are used to attack information systems

phishing ☆ a technique for attempting to acquire sensitive data, such as bank account numbers, through a fraudulent solicitation in email or on a Web site, in which the perpetrator masquerades as a legitimate business or reputable person

pharming ✤ attack aimed at redirecting a website's traffic to another, bogus website

malware any software including malicious code

virus malware which, if run by the user, causes damages in the system; it is not able to propagate to other systems, unless the user transfers, even involuntarily, the infected file

worm malware which saturates CPU, memory or network resources by creating replications of itself, and it is able to propagate to other systems without user intervention

Trojan horse ✡ the means through which malwares are usually distributed

backdoor ✡ unauthorized access point which can be used by a malware to bypass the authentication system and enter the system

rootkit ✡ set of tools, hidden in the system, to gain system access with administrative privileges (e.g. modified program, library, driver, kernel module, hypervisor)

A.2 Basics of ICT security

cryptography ✪ the discipline which embodies principles, means, and methods for the transformation of data in order to hide its information content [. . .]

encryption ✪ the cryptographic transformation of data to produce ciphertext

cipher ☆ a cryptographic algorithm for encryption and decryption

clear text ✪ intelligible data, the semantic content of which is available

cipher text ✪ data produced through the use of encipherment; the semantic content of the resulting data is not available

key ☆ an input parameter used to vary a transformation function performed by a cryptographic algorithm

symmetric cryptography ☆ a branch of cryptography in which the algorithms use the same key for both of two counterpart cryptographic operations (e.g., encryption and decryption)

symmetric key ☆ a cryptographic key that is used in a symmetric cryptographic algorithm

asymmetric cryptography ☆ a modern branch of cryptography (popularly known as "public-key cryptography") in which the algorithms use a pair of keys (a public key and a private key) and use a different component of the pair for each of two counterpart cryptographic operations (e.g., encryption and decryption, or signature creation and signature verification)

asymmetric key ☆ a cryptographic key that is used in an asymmetric cryptographic algorithm

key pair ☆ a set of mathematically related keys – a public key and a private key – that are used for asymmetric cryptography and are generated in a way that makes it computationally infeasible to derive the private key from knowledge of the public key

public key ☆ the publicly disclosable component of a pair of cryptographic keys used for asymmetric cryptography

private key ☆ the secret component of a pair of cryptographic keys used for asymmetric cryptography

strong ☆ used to describe a cryptographic algorithm that would require a large amount of computational power to defeat it

key length ☆ the number of symbols (usually stated as a number of bits) needed to be able to represent any of the possible values of a cryptographic key

effective key length ✯ a measure of strength of a cryptographic algorithm, regardless of actual key length

security by obscurity ☆ attempting to maintain or increase security of a system by keeping secret the design or construction of a security mechanism

key establishment ☆ a procedure that combines the key-generation and key-distribution steps needed to set up or install a secure communication association

key generation ☆ a process that creates the sequence of symbols that comprise a cryptographic key

key distribution ☆ a process that delivers a cryptographic key from the location where it is generated to the locations where it is used in a cryptographic algorithm

out-of-band ☆ information transfer using a channel or method that is outside (i.e., separate from or different from) the main channel or normal method

key transport ☆ a key establishment method by which a secret key is generated by a system entity in a communication association and securely sent to another entity in the association

key agreement ✢ a method for negotiating a key value on line without transferring the key, even in an encrypted form, e.g., the Diffie-Hellman technique

block cipher ☆ an encryption algorithm that breaks plain text into fixed-size segments and uses the same key to transform each plaintext segment into a fixed-size segment of cipher text

mode of operation ☆ a technique for enhancing the effect of a cryptographic algorithm or adapting the algorithm for an application, such as applying a block cipher to a sequence of data blocks or a data stream

known-plaintext attack ☆ a cryptanalysis technique in which the analyst tries to determine the key from knowledge of some plaintext-ciphertext pairs (although the analyst may also have other clues, such as knowing the cryptographic algorithm)

initialization value (IV) ☆ an input parameter that sets the starting state of a cryptographic algorithm or mode

liveness ☆ a property of a communication association or a feature of a communication protocol that provides assurance to the recipient of data that the data is being freshly transmitted by its originator, i.e., that the data is not being replayed, by either the originator or a third party, from a previous transmission

nonce ☆ a random or non-repeating value that is included in data exchanged by a protocol, usually for the purpose of guaranteeing liveness and thus detecting and protecting against replay attacks

stream cipher ✯ an encryption algorithm that breaks plain text into a stream of successive elements (usually, bits) and encrypts the n-th plaintext element with the n-th element of a parallel key stream, thus converting the plaintext stream into a ciphertext stream

one-time pad ☆ an encryption algorithm in which the key is a random sequence of symbols and each symbol is used for encryption only one time – i.e., used to encrypt only one plaintext symbol and thus produce only one ciphertext symbol – and a copy of the key is used similarly for decryption

pseudorandom ☆ a sequence of values that appears to be random (i.e., unpredictable) but is actually generated by a deterministic algorithm

seed ☆ a value that is an input to a pseudorandom number generator

elliptic curve cryptography (ECC) ★ a type of asymmetric cryptography based on mathematics of groups that are defined by the points on a curve, where the curve is defined by a quadratic equation in a finite field

hash function ☆ a function H that maps an arbitrary, variable-length bit string, s, into a fixed-length string, $h = H(s)$ [...]

digest ✣ a sort of fixed-length 'summary' of data, usually computed by means of a dedicated (= for security purpose) cryptographic hash algorithm

cryptosystem the set of a cryptographic algorithm and a hash algorithm used to guarantee the security of a certain system

keyed hash ☆ a cryptographic hash in which the mapping to a hash result is varied by a second input parameter that is a cryptographic key

digital signature ☆ a value computed with a cryptographic algorithm and associated with a data object in such a way that any recipient of the data can use the signature to verify the data's origin and integrity

authenticated encryption mechanism ✣ cryptographic technique used to protect the confidentiality and guarantee the origin and integrity of data, and which consists of two component processes: an encryption algorithm and a decryption algorithm

key escrow ✣ possibility to recover a key even without the owner's consent

key recovery ✦ mechanisms and processes that allow authorized parties to retrieve the cryptographic key used for data confidentiality

A.3 The X.509 standard, PKI and electronic documents

public key certificate ✣ a data structure used to securely bind a public key to some attributes

public-key infrastructure (PKI) ✣ the technical and administrative infrastructure put in place for the creation, distribution and revocation of public key certificates

certification authority (CA) ☆ an entity that issues digital certificates (especially X.509 certificates) and vouches for the binding between the data items in a certificate

registration authority (RA) ☆ an optional PKI entity (separate from the CAs) that does not sign either digital certificates or CRLs but has responsibility for recording or verifying some or all of the information (particularly the identities of subjects) needed by a CA to issue certificates and CRLs and to perform other certificate management functions

end entity ☆ a system entity that is the subject of a public-key certificate and that is using, or is permitted and able to use, the matching private key only for purposes other than signing a digital certificate; i.e., an entity that is not a CA

relying party ☆ a system entity that depends on the validity of information (such as another entity's public key value) provided by a digital certificate

certification hierarchy ☆ a tree-structured (loop-free) topology of relationships between CAs and the entities to whom the CAs issue public-key certificates

root CA ☆ the CA that is the highest level (most trusted) CA in a certification hierarchy; i.e., the authority upon whose public key all certificate users base their validation of certificates, CRLs, certification paths, and other constructs

self-signed certificate ☆ a public-key certificate for which the public key bound by the certificate and the private key used to sign the certificate are components of the same key pair, which belongs to the signer

issue ☆ generate and sign a digital certificate (or a CRL) and, usually, distribute it and make it available to potential certificate users (or CRL users)

subject ☆ the name (of a system entity) that is bound to the data items in a digital certificate; e.g., a DN that is bound to a key in a public-key certificate

issuer ☆ the CA that signs a digital certificate or CRL

repository ☆ a system for storing and distributing digital certificates and related information (including CRLs, CPSs, and certificate policies) to certificate users

certificate revocation ☆ the event that occurs when a CA declares that a previously valid digital certificate issued by that CA has become invalid; usually stated with an effective date

certificate revocation list (CRL) ☆ a data structure that enumerates digital certificates that have been invalidated by their issuer prior to when they were scheduled to expire

delta CRL ☆ a partial CRL that only contains entries for certificates that have been revoked since the issuance of a prior, base CRL

indirect certificate revocation list (ICRL) ☆ in X.509, a CRL that may contain certificate revocation notifications for certificates issued by CAs other than the issuer (i.e., signer) of the ICRL

certificate status responder ☆ a trusted online server that acts for a CA to provide authenticated certificate status information to certificate users; offers an alternative to issuing a CRL

time stamp ☆ with respect to a data object, a label or marking in which is recorded the time (time of day or other instant of elapsed time) at which the label or marking was affixed to the data object

personal security environment (PSE) ✧ secure local storage for an entity's private key, the directly trusted CA key and possibly other data; depending on the security policy of the entity or the system requirements, this may be, for example, a cryptographically protected file or a tamper resistant hardware token

cryptographic token ☆ a portable, user-controlled, physical device (e.g., smart card or PCMCIA card) used to store cryptographic information and possibly also perform cryptographic functions

cryptographic smart card ★ chip card with memory and autonomous cryptographic capability

proof-of-possession ☆ a protocol whereby a system entity proves to another that it possesses and controls a cryptographic key or other secret information

digital envelope ☆ a combination of

(a) encrypted content data (of any kind) intended for a recipient

(b) the content encryption key in an encrypted form that has been prepared for the use by the recipient

electronic signature ★ the set of data in an electronic form which is attached or logically bound to other data in an electronic form and provides its means of authentication

qualified certificate ✯ a public-key certificate that has the primary purpose of identifying a person with a high level of assurance, where the certificate meets some qualification requirements defined by an applicable legal framework, such as the European Directive on Electronic Signature

A.4 Authentication systems

authentication ☆ the process of verifying a claim that a system entity or system resource has a certain attribute value

identification ☆ an act or process that presents an identifier to a system so that the system can recognize a system entity and distinguish it from other entities

verification ☆ the process of examining information to establish the truth of a claimed fact or value

dictionary attack ☆ an attack that uses a brute-force technique of successively trying all the words in some large, exhaustive list

ransomware ✯ a type of malware which restricts access to the computer system that it infects, and demands a ransom paid to the creator(s) of the malware in order for the restriction to be removed

salt ☆ a data value used to vary the results of a computation in a security mechanism, so that an exposed computational result from one instance of applying the mechanism cannot be reused by an attacker in another instance

challenge-response ☆ an authentication process that verifies an identity by requiring correct authentication information to be provided in response to a challenge [...]

one-time password ☆ a simple authentication technique in which each password is used only once as authentication information that verifies an identity [...]

token ☆ a data object or a physical device used to verify an identity in an authentication process

biometric authentication ☆ a method of generating authentication information for a person by digitizing measurements of a physical or behavioral characteristic [...]

Single Sign-On (SSO) ✱ providing the user with a single 'credential' by which to authenticate himself for all operations on any system

A.5 Security of IP networks

accounting ✲ the process of tracking and/or analyzing user activities on a network by logging key data (e.g. amount of time in the network, services accessed, amount of data transferred)

Virtual Private Network (VPN) ✳ a hardware and/or software technique to create a private network while using shared or anyway untrusted channels and transmission devices

A.6 Firewall and IDS/IPS

firewall ☆ an internetwork gateway that restricts data communication traffic to and from one of the connected networks (the one said to be "inside" the firewall) and thus protects that network's system resources against threats from the other network (the one that is said to be "outside" the firewall)

bastion host ☆ a strongly protected computer that is in a network protected by a firewall (or is part of a firewall) and is the only host (or one of only a few) in the network that can be directly accessed from networks on the other side of the firewall

buffer zone ☆ a neutral internetwork segment used to connect other segments that each operate under a different security policy

honey pot ☆ a system (e.g., a web server) or system resource (e.g., a file on a server) that is designed to be attractive to potential crackers and intruders, like honey is attractive to bears

screening router ☆ an internetwork router that selectively prevents the passage of data packets according to a security policy

intrusion detection ☆ sensing and analyzing system events for the purpose of noticing (i.e., becoming aware of) attempts to access system resources in an unauthorized manner

Intrusion Detection System (IDS) ✶ system to identify individuals using a computer or a network without authorization

Intrusion Prevention System (IPS) ✦ system that can detect an intrusive activity and can also attempt to stop the activity, ideally before it reaches its targets

Sources

Course slides

✡ *Introduction to the security of ICT systems* © Antonio Lioy - Politecnico di Torino (2005-2014)

✣ *Basics of ICT security* © Antonio Lioy - Politecnico di Torino (2005-2014)

★ *The X.509 standard, PKI and electronic documents* © Antonio Lioy - Politecnico di Torino (1997-2011)

✱ *Authentication systems* © Antonio Lioy - Politecnico di Torino (1995-2014)

✳ *Security of IP networks* © Antonio Lioy - Politecnico di Torino (1997-2014)

✶ *Firewall and IDS/IPS* © Marco Domenico Aime, Antonio Lioy - Politecnico di Torino (1995-2014)

Requests for Comments

☆ RFC 4949: R. Shirey, *Internet Security Glossary, Version 2* © The IETF Trust (2007)

✯ RFC 3739: S. Santesson, M. Nystrom, T. Polk, *Internet X.509 Public Key Instrastructure: Qualified Certificates Profile* © The Internet Society (2004)

ISO International Standards

✪ ISO 7498-2:1989: *Information processing systems – Open Systems Interconnection – Basic Reference Model – Part 2: Security Architecture* © 1989 ISO

✢ ISO/IEC 9594-8:1998: *Information Technology – Open Systems Interconnection – The Directory: Authentication framework* © 1998 ISO/IEC

✧ ISO/IEC 15945:2002: *Information technology – Security techniques – Specification of TTP services to support the application of digital signatures* © 2002 ISO/IEC

✣ ISO/IEC 19772:2009: *Information technology – Security techniques – Authenticated encryption* © 2009 ISO/IEC

✤ ISO/IEC 29115:2013: *Information technology – Security techniques – Entity authentication assurance framework* © 2013 ISO/IEC

Other

☆ G. Huff, *Trusted Computer Systems – Glossary*, MTR 8201, The MITRE Corporation, March 1981

★ *2014 Data Breach Investigations Report* (DBIR) © 2014 Verizon

✯ B. Schneier, *Applied Cryptography Second Edition*, John Wiley & Sons, Inc., New York, 1996

✰ NSA, *Information Assurance Technical Framework*, Release 3, NSA, September 2000

✦ CNSSI-4009: The Committee on National Security Systems Instruction No 4009 *National Information Assurance Glossary* © NCSC

✫ Wikipedia

✲ Microsoft Language Portal © Microsoft