POLITECNICO DI TORINO

Information Systems Security

Notes from the course 01TYMOV of Prof. Antonio Lioy

A.A. 2020/21

Author: Marco Smorti
Auditor: Riccardo Zaccone
Contents updated to lesson: 20 (11/12/2020)
Review of contents updated to lesson: 20 (11/12/2020)
Last modified on: 20/01/2021 13:13:00

Summary

Introduction to the security of ICT systems

Cybersecurity nowadays became very important: since every system is built on computer systems any kind of damage will nearly cost an economic lost, and this is the major problem. Even indirect attacks that does not mean to steal money, they have economic costs. Cybersecurity is important even because the attack can be performed without going physically to the target place.

The reason of why Cybersecurity is important are:

- Big damages on successful attacks;
- Easy accessibility of systems;

We must consider all the possible consequences of a successful attack. First, there can by **financial loss** (*direct loss* if for example I get the credentials of bank accounts and *indirect loss* if the information that the company has been attacked make negative effects on the stock exchange). There can be **recovery cost** because every successful attack is a damage, and there will be costs to set the system back to normal operations and to improve the system to avoid new attacks. There can be **productivity loss** if the attacks stop or delay processes. A successful attack may get a **business disruption** because customers may look to alternative suppliers if a company is easy to attack.

For all these reasons we should protect systems. Most of the innovation nowadays is based on two main pillars:

- The fact that we can always communicate from each part of the world (communication networks);
- The increasing use of personal and mobile devices;

These two pillars are no more enough for an innovation product: each new product needs a security system.

Complexity of the ICT scenario

The ICT scenario is complex for many reasons. For example, the big amount of **different** mobile and connected **devices**: desktop, laptop, tablet, smartphone, smart TV, fridge, car. All these things can now communicate through internet, so the need to be secured. **Communication networks** have also become data-only networks (which means that there is no more an analog phone networks), and everything goes through the network. This means that everything is attackable. Not only wireless networks can be attacked, but also cabled networks are highly attackable and insecure. **Distributed services** are now increasing, and it means that we often are called to find technical solutions to sustain them, outsourcing some parts of the server, hosting, farming and finally use cloud. This means that computers are no more inside a company but somewhere else, so you need to also trust the provider who hosts your services. Also, programming is becoming increasingly complex as consequence of software stratification, integration of frameworks, language mixes, and this means that easy easier to make mistakes.

For what concerns security, these motivations can be summed up in the **first axiom of engineering**:

> "*The more complex a system is, the more difficult its correctness verification will be.*"

This means we must keep a system as easier as possible. For example, the number bugs in a program is more than quadratic to its number of lines of code and the complexity of the current information systems is in favor of the attackers, who can find attack paths increasingly ingenious and unforeseen by the defenders. A way to express this concept **the KISS rule**: "*Keep it Simple, Stupid*".

A definition of ICT security

Each of us has a different concept of security: for example, the obligation to use safety belts when driving depends on country to country. Security is a personal concept but at engineering we need to give some formal

definitions. Cybersecurity is a distributed part of a company and each employee of a company must have the awareness for cybersecurity.

1. "*Cybersecurity is the set of products, services, organization rules and individual behaviors that protect the ICT system of a company*."

Let us explain the keywords in the definition.

- **Products**: refers to something that people can buy (such as products for firewall and VPN);
- **Services**: these services are implemented by buying products;
- **Organization rules**: they are required because even if for example a new system is set up with passwords, rules must give information to employees on how much complex the password must be, otherwise there will be no rules but personal behaviors, which could make the use of technical solutions less effective;

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

More in detail, these three concepts have the following meaning:

- **Confidentiality** covers two related concepts:
    - **Data confidentiality**: assures that private or confidential information is not made available or disclosed to unauthorized individuals;
    - **Privacy**: Assures that individuals control or influence what information related to them may be collected and stored and by whom and to whom that information may be disclosed.

  In terms of requirements and the definition of a loss of security, it means preserving authorized restrictions on information access and disclosure, including means for protecting personal privacy and proprietary information. A loss of confidentiality is the unauthorized disclosure of information.

- **Integrity** covers two related concepts:
    - **Data integrity**: assures that information (both stored and in transmitted packets) and programs are changed only in a specified and authorized manner;
    - **System integrity**: assures that a system performs its intended function in an unimpaired manner, free from deliberate or inadvertent unauthorized manipulation of the system.

  In terms of requirements and the definition of a loss of security, it means guarding against improper information modification or destruction, including ensuring information nonrepudiation and authenticity. A loss of integrity is the unauthorized modification or destruction of information.

- **Availability**: assures that systems work promptly, and service is not denied to authorized users. In terms of requirements and the definition of a loss of security, it means ensuring timely and reliable access to and use of information. A loss of availability is the disruption of access to or use of information or an information system.

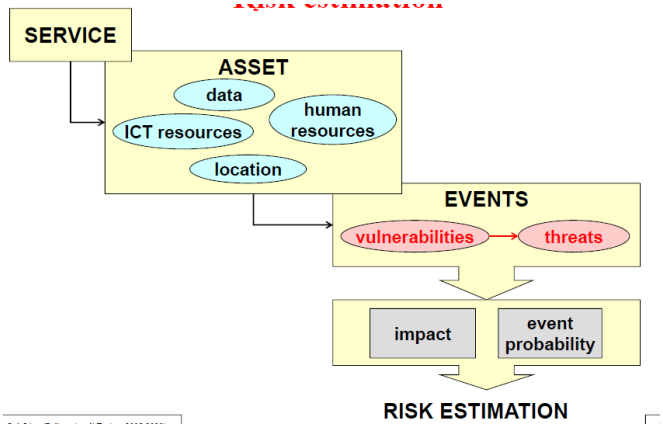
This definition is a good starting point for cybersecurity, but more than the C.I.A. triad is required; two of the most mentioned are as follows:

- **Authenticity**: the property of being genuine and being able to be verified and trusted; confidence in the validity of a transmission, a message, or message originator. This means verifying that users are who they say they are and that each input arriving at the system came from a trusted source.
- **Accountability**: the security goal that generates the requirement for actions of an entity to be traced uniquely to that entity. This supports nonrepudiation, deterrence, fault isolation, intrusion detection and prevention, and after-action recovery and legal action.

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

3. "*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.*"

Risk estimation



Before setting up a defense, we must understand what the risks are. To make a risk estimation it is good to start from the **service**. Once we know the service we need to protect, we must identify which are the assets used to provide that service and there are 4 categories of assets: **ICT resources** (computer, disks, networks), **data** (not the disks but something intangible that could be deleted or modified), **location** (assets must be inside a protected room), **human resources** (means the close amount of people that have the knowledge that must not be shared). After thinking about assets, the next step are the events that could affect their normal operation. The first point is that each asset has some **vulnerabilities** (for example disks, that are vulnerable to hammer hitting the disk) and some vulnerabilities can be a real **threat** depending on the environment. For example, if the disk is left in an open place maybe someone could use the hammer to crash the disk, but if the disk is locked in a room where nobody accesses the vulnerability still exists but is not a real threat.

So, the process of analyzing a service searching for risks take place as follows:

- Find the **assets** of the service to be protected;
- Finding the **vulnerabilities** of each asset;
- Finding the **treats**, giving the way in which the assets are used;

Once we identify the threats we must:

- decide for each threat which **impact** it could have (what happens if disk is destroyed? If there's only one copy it could be a disaster, but if there are many copies that is not a problem);
- the **event probability** of the threat. This is the last point to get the **risk estimation**.

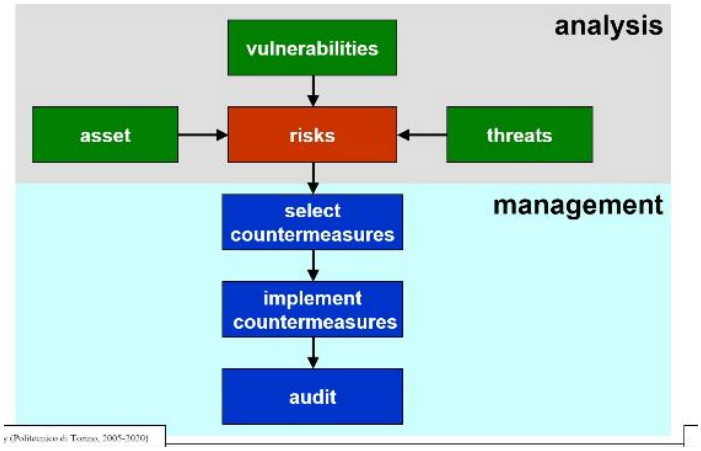
At the end of the process we have a list of items with the following format:

Name of the risk	Impact (e.g. low, medium, high)	Probability

Recap of terminology:

- **Asset:** the set of goods, data and people needed for an IT service;
- **Vulnerability**: weakness of an asset;
- **Threat:** deliberate action/accidental event that can produce the loss of a security property exploiting a vulnerability;
- **Attack:** threat occurrence (deliberate action);
- **(negative) Event:** threat occurrence (accidental event);

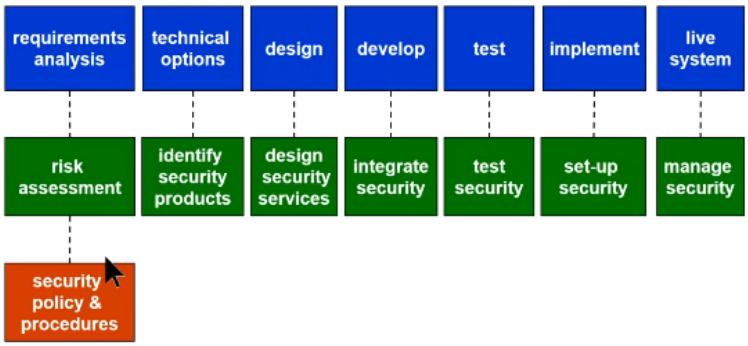
Analysis and management of security



Risk estimation is only a piece for analysis and *management of security*. We can see that **assets, vulnerabilities, threats** give us the **risks** and then we start the **management** of the security. This means that for those risks that are not acceptable, either because they have high impact or high probability to end up in an attack, we need to select countermeasures and implement them. The last step is **audit,** that means that some independent person comes to check our work (if we correctly identified risks, selected the correct countermeasures, implemented them correctly).

In this course we will consider three of these blocks: vulnerabilities, the available countermeasures and how to implement countermeasures.

What is the correct step in the lifecycle of a system where to implement security? Brief answer: there is not such correct point where to implement security, because it must be taken care of in **each step** of the design.



More in detail, when we perform the **analysis of requirements** for our system we must perform risk assessment: basing on these risks we can define **security policy & procedures** that we will apply for the rest of the design of the system.

When evaluating **technical options,** we need also to identify the security products. For example, when we evaluate which database to use, among other characteristics such as speed and cost, we need also to consider security. If we choose a database that automatically encrypts data, we have already solved a security issue. On the contrary if we select a faster database but that does not provide any encryption system, we will need to design that separately with additional cost and effort.

When **designing** the services that system will offer, we need to also design the security services part. We must add security in each step of the design and not "later", because if we make a prototype website or app without any security system it will be difficult to add security systems later on.

Figure 1 Example comic that explains reality of systems

During the **development** of our system, we must integrate security in each step and to check that the design is correctly implemented we must **test the system** and **test security**: an example of this is testing against unexpected inputs, to be sure that the system does not accept and process wrong inputs.

While **implementing** our system it is important to set-up security mechanisms: there are several systems that include security functions, but they are deactivated, because they have a cost (generally activating security the system gets slower). For example, the default password of a home router could be shared among different units, so it is needed to change that password.

When the **system operation**, security must be managed day by day, because security scenario changes every day.

Relations in the security field

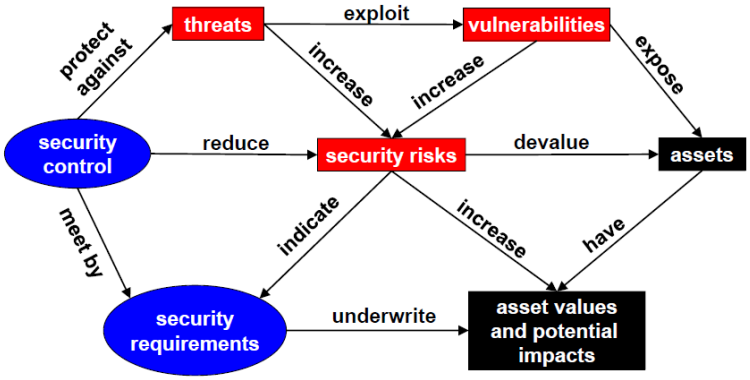


Figure 2 Relations in the security field

The black box is the system itself. Assets are exposed to vulnerabilities and those vulnerabilities increase security risks. As well as threats exploits vulnerabilities, the existence of vulnerabilities gives the opportunity to create a threat.

On the left there are the security requirements that we want to implement. The security requirements are indicated by security risks and security requirements are met by security control. The **security control** is the most important piece of the picture nowadays: it is an element placed in the system that protect against one specific threat and it reduces risks to which system is exposed to. Examples of security control are firewall, VPN, and disk encryption.

Some terminology

- **Incident:** security event that compromises the integrity, confidentiality, or availability of an information asset (generic definition)
- **(data) breach:** incident that results in the disclosure or potential exposure of data;
  - disclosure: I give data to someone;
  - exposure: data are available for anybody who can know where they are;
- **(data) disclosure:** a breach for which it was confirmed that data was actually disclosed (not just exposed) to an unauthorized party.

The difference between the last two is that the last one is a breach into data that were not disclosed.

Window of exposure

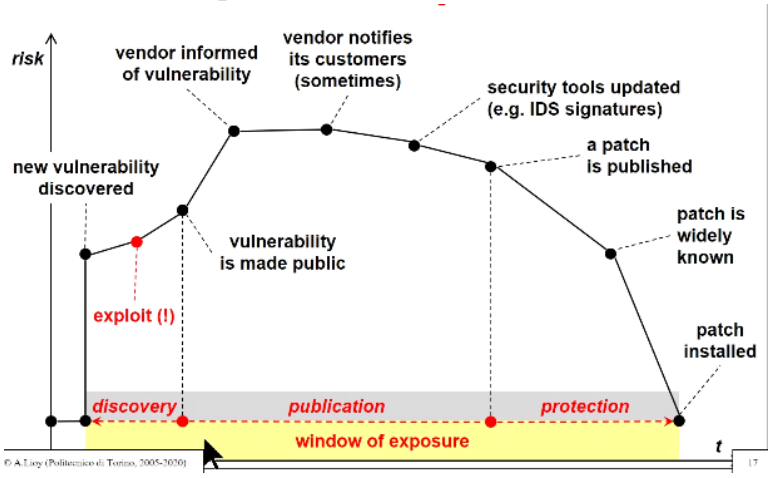


Figure 3 The three phases of the window of exposure (WOE): discovery, publication and protection

The **window of exposure** is the time between the discovery of a new vulnerability and the installing of a patch. Reading the graphic from the lest we see that there is a low level of risk and it can never be 0, but something close to 0. At some point a new vulnerability is discovered and risk goes high (because if it is new, we cannot stop it). At some point (the red point) someone is exploiting the vulnerability to perform an attack. In this way the vulnerability is made public and everybody can know about it. This first phase is named **discovery**.

After the attack is being performed there are two categories of people informed of this: the bad guys (who wants to attack the system) and the good guys (who try to protect the system). Among these two categories there is also the vendor of the product that must be informed of vulnerability, and vendor should notify its customers of new vulnerability. While the vendor is working to fix the vulnerability, users of that product should not try to fix it (typically impossible) but they should update their security tools at least to detect if the vulnerability is being used actively for an attack.

This is the phase of **publication** in which the vulnerability is known to the public and everybody is waiting for a fix and trying at least to detect attacks. Finally, at some point in time vendor is creating a patch that fix vulnerability, but patch must be distributed, and the risk goes down only when the patch is widely known and finally installed. The window of exposure can last days or even months: therefore, security is a work that never terminates.

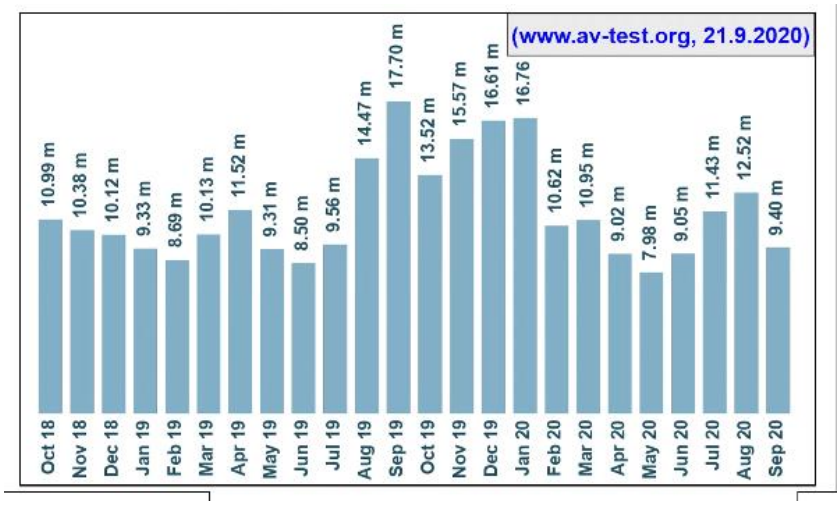


Figure 4 Number of attacks to system per month using malwares

The graphic shows that there are about 10 million attacks per month using malwares. That is why we should always get our antivirus/antimalware updated.

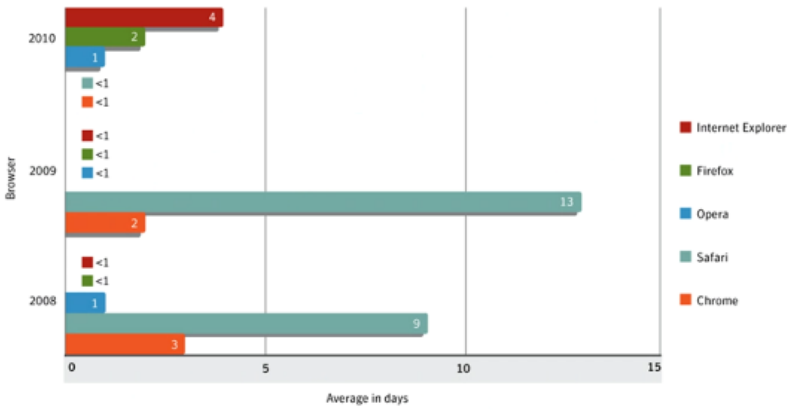


Figure 6 WOE: average value for browsers in the period 2008-10

Statistics of the value of window of exposure for browsers. The worst thing was in 2009 when Safari had an exposure for 13 days before Apple was able to provide a fix. Also, in 2008 it was 9. In 2010 IE with 4 average in days.

Figure 7 WOE: server web (2010)

If we consider web server from all sectors, the 44% of servers was always vulnerable to an attack in each day of a year.

The **0 day** is literally the first day when the vulnerability is reported to public (and has not been fixed yet).

The banking sector is the one with the best "green bar" (rarely vulnerable, 30 days or less a year).

In general, it is difficult to have security. Vulnerabilities can be discovered from both categories: good guys and bad guys. There are people that investigate on software to discover vulnerabilities and inform the vendor of it. This is done to make patch fix before the vulnerability is discovered from bad guys. The "**0-day initiative**" (ZDI) discovers vulnerability and inform the society **before** making it public. The ZDI gives 120 days from the discovery of a new vulnerability to the vendor to fix the vulnerability before making it public. Longer deadlines can be risky, because even bad guy can discover the vulnerability in the meanwhile.

Example:

- *8 May 18*: ZDI reports the vulnerability to the vendor and he acknowledge about the report.
- *14 May 18*: the vendor replies that they successfully reproduced the issue reported from ZDI
- *9 Sep 18*: the vendor reported an issue with the fix and that the fix *might not* make the September release
- *10 Sep 18*: ZDI cautioned potential 0-day (which means that the vulnerability which fix is not available is going to being published)
- *11 Sep 18*: the vendor confirmed that the fix did not make the build
- *12 Sep 18*: ZDI confirmed intention to 0-day on 20 Sep 18.

What is security?

"*Security is a process, not a product*."

(Bruce Schneier, Crypto-Gram, May 2005)

If we have learned anything from the past couple of years, it is that **computer security flaws are inevitable**. Systems break, vulnerabilities are reported in the press, and still many people put their faith in the next product, or the next upgrade, or the next patch. "This time it's secure," they say. So far, it has not been.

Security is a process, not a product. Products provide some protection, but the only way to effectively do business in an insecure world is to put processes in place that recognize the inherent insecurity in the products. **The trick is to reduce your risk of exposure regardless of the products or patches**.

To make it possible it is needed to follow some security principles.

Security principles

- **Security in depth**: if the enemy can defeat the first line, there must be a second line to stop the attacker. Do not rely on just one defense, because that defense may have a bug or problem. It is better to have multiple levels of defense, so as the attacker breaks through the defenses, it will become increasingly difficult to keep penetrating;
- **Security by design:** it means that the design of the security is made along with the system and not at the end;
- **Security by default**: users should not have the choice to activate security or not. Security should be ON by default and it should be a big effort to delete security from the system.
- **Least privilege**: it means that any element which is operating inside the system should be assigned with the minimum amount of privileges that permits it to perform its task. Imagine that we have huge privilege and computer is being attacked from a virus: the virus could get access to everything because we have full privilege on that computer;
- **Need-to-know**: it means that we should give access to any component of the system only to data that are required to execute that task. Let us consider Amazon: several people who work inside. When you make an order on Amazon there is a 1st person that is taking the order. From the order he can only see what you ordered but the person does not know who ordered and the destination of the goods.

European Central Bank

ECB made on 31/01/2013 some "*Recommendations for the security of Internet payments*". In general, when there is a regulation there are **generic** recommendations, but typically lawyers do not want to get into details. In the security scenario this is changed recently. It is not possible to let individual company to decide what are the good rules for security. These recommendations apply to payment schemas governance authorities, to payment service providers (PSP), merchants that use credit cards for payments.

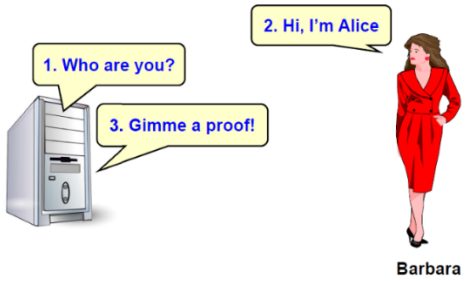
The main recommendations are:

- Protect the initiation of Internet payments, as well as access to sensitive payment data, by **strong customer authentication**;
- **Limit the number of log-in or authentication attempts**, define rules for Internet payment services session "time out" and set time limits for the validity of authentication;
- Establish **transaction monitoring mechanisms** designed to prevent, detect, and block fraudulent payment transactions; (*reminds Security in depth*)

- **Provide assistance** and guidance to customers about best online security practices, set up alerts and provide tools to help customers monitor transactions.

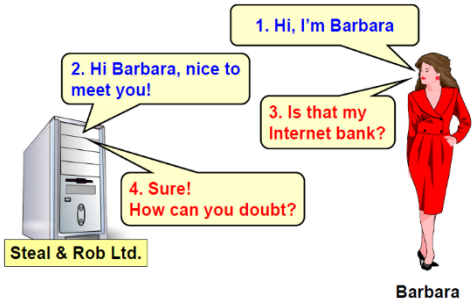
Security properties

- Authentication (simple/mutual);
- Peer authentication;
- Data/origin authentication;
- Authorization, access control;
- Integrity;
- Confidentiality, privacy, secrecy;
- Non-repudiation;
- Availability;
- Traceability, accountability.



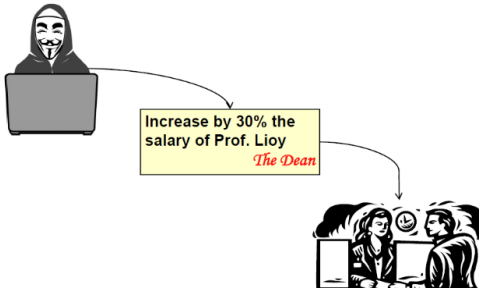
Peer authentication (simple)

When there is a communication between two peers that take part into the communication they must authenticate. Two entities are considered peers if they implement to same protocol in different systems; for example, two TCP modules in two communicating systems. Computer system usually asks some questions like the ones in the picture. Typically for a proof we give a password. This is the **simple authentication**: only one part authenticates. It attempts to provide confidence that an entity is not performing either a masquerade or an unauthorized replay of a previous connection.



Mutual peer authentication

In mutual peer authentication we need also formal proof that we are connected to the real server. A fake server could display the same page of the ordinary bank just to get credentials from user. When we connect to a server, we typically trust the server, but we need a proof. Both parts authenticate with each other.



Data authentication

Someone could write an email to increase salary of someone, signed by the director, but there are no proof that the email came from the director. An electronic email has no proper authentication. The same happens for files. Data are normally not authenticated, so we could need a system of authentication also for data.

Non-repudiation

Non-repudiation is a formal proof, which is acceptable by a court of justice, that gives *undeniable* evidence of the data creator. Nonrepudiation prevents either sender or receiver from denying a transmitted message. This has several implications because we do not only need authentication, but also **integrity**, because if someone changes data from a document that should be detect because it is no more the original authenticated document. There is a **difference between authentication and identification**: authentication means that we used some electronics mean to prove the identity (for example username and password), but what if password has been

stolen? Is that really you or someone else? On the contrary, identification is much stronger, for example the touch ID on smartphone, because the user is the only one who can perform that operation.

Example of non-repudiation

Let us consider non-repudiation of an electronic signature:

- Syntax (is that your signature?)
- Semantics (did you understand what you were signing?) - what you don't understand has no legal value (for example the small lines in a document that are not understandable).
- Will (have you signed voluntarily?)
- Identification (was really YOU the signer?)
- Time (when did you sign?)
- Place (where did you sign?)

The electronic signature is a set of bits that represent the signature of a person. If we do not know who these bits must be placed to represent the signature, we cannot know of which person the signature is. So, for these reasons there are specifications for electronic signature.

Availability

An availability service is one that protects a system to ensure its availability. This service addresses the security concerns raised by denial-of-service attacks. It depends on proper management and control of system resources and thus depends on access control service and other security services.

Authorization (access control)

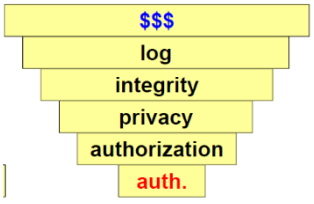
Authentication means that we identify who are the actors in the system. After authentication we can take decisions. For example, in the figure Barbara has identified herself correctly and then is asking to the computer "open the box of Alice's car" and the computer is performing an **authorization decision (or access control)**. This is because the car in the box is not Barbara's car. So, there is also a system that must know if someone is authorized to perform an operation or not.

Important difference!

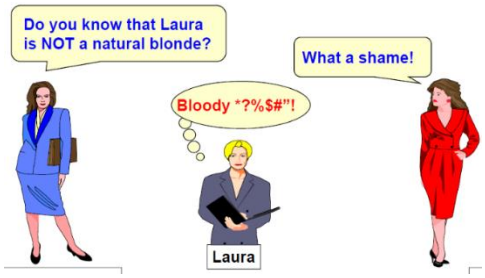
- The **authorization** is the check if you, who has been already authenticated, are authorized to perform an operation or not;
- The **authentication** is the identification of the users of a system.

Pyramid of security

Authentication is the most basic feature in our system: if we do not identify the actors we cannot get on the top of the pyramid. Based on authentication there is: authentication (who can do what), privacy (who can read), integrity (who can modify), log (take notes of actions).



Unfortunately, most of the systems have a very weak authentication based on username and password and on top of that you build have a very complex system with a huge value. If authentications get broken, all the other security mechanisms are compromised. Often is what is found in companies because managers do not understand the technical principles needed to realize how these different security mechanisms are to be taken in place.



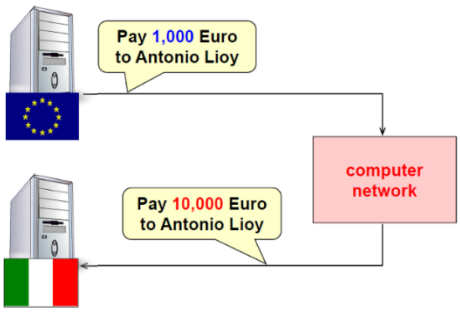
Privacy (communication)

Privacy has several meanings: communication privacy refers, for example, when 2 peers are communicating it would like to be not possible for a 3rd one to understand the communication.

Even if someone can read the data passing through the network, no one (except the two end users) should be able to understand what's inside the communication.

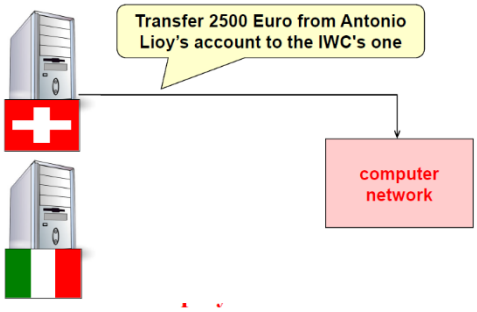
Privacy (data, actions, position)

- **Data privacy**: even if we have access to the place where data are stored, we might not be able to access the data;
- **Privacy of actions**: companies have the rights to trace the websites visited from users, as well as police according to the law which is anti-terrorism. All the data sent in Italy are stored for 7 years in case of future investigations;
- **Privacy of position**: these kinds of information are available for who manage the network. While defining the security properties of an application we must be worried about data, actions performed over the network and eventually the position from which the communication was made.



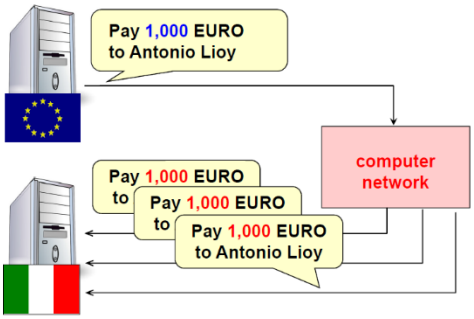
Integrity (data modification)

It means that if data have been modified, we are able to detect it. It doesn't mean that nobody can change the data: that's impossible. Network manager can always read and eventually modify data, so that's why integrity refers to the detection of modified data.



Integrity (data cancellation/filtering)

Data can also be deleted, so we must ensure that if data cancellation happens, we are detecting that. This is more difficult to detect, because the receiver does not receive anything (and do not have hint that the payment in the picture should be received).



Reply attack

Data sent on the network can be encrypted and no more modifiable from the network manager. But it is possible to record the message sent to the network and reply it multiple times. The authentication will pass because the message is not modified, and it is not easily understood by developers.

Data protection

During the explanation of security properties, we always talked about protecting data. There are two type of data protection:

- **Data in transit**: when data are transmitted over a communication channel;
- **Data at rest**: when data are stored in a memory device.

Where is the enemy?

To defend something, we must know where the enemy is. There are a few possibilities:

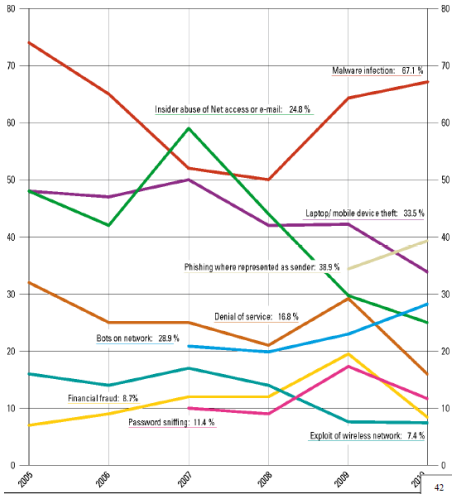
- **Outside our organization:** In this case a perimeter (or boundary) defense called *firewall* is required.
- **Outside our organization, except for our partners**: in this case the firewall needs a supplement with some paths or roots that are protected and permit the communication between trusted users. In other words, is required an extranet protection VPN, which is an intranet extended to include also partners.
- **Inside our organization:** In this case we should protect the Local Area Network (LAN) and the intranet applications, which is quite difficult since we need to share information between users in the same network.
- **Everywhere**: since the attacker can be inside or outside the organization, it can be everywhere, because we can connect to the organization from each part of the world nowadays. In this case **security must be placed at the application level**. Secondly since applications deal with data, these must be protected independently from the place where they are stored. For example, if Dropbox is used but is not considered secure by the company, it is possible to encrypt the data before uploading them on Dropbox.

Attack origin (2016)

Statistics are difficult to be done because companies do not disclose that they have been attacked, so statistics are not often updated. The percentage of external/internal attacks from a statistic of Verizon is:

- Internal 20%
- External 80%

Whenever reading statistics we must be careful about who were the people of the companies interviewed. Verizon is one of the biggest internet providers of the world, so their customers will be users connected to internet, and most of the attacks come from internet. In case statistics came from local attacks like manufactures industries, statistics would have been different.



The type of attack changes over the time. The graphic on the left shows that trend of attacks number every year, but malware infection is the most popular. This is because even there are antivirus, because of the number of new malwares developed every month there is always the chance that a malware exploits an unknow vulnerability (see Window of exposure).

From the graph we notice also that laptop or mobile device theft is quite frequent. The problem of stolen laptop/smartphone is not just the economy loss to replace the stolen device but also the loss of data that become unavailable (in case the user had not a backup) and the spreading of restricted information inside the devices.

An example is the article from Corriere on 2009 that says: "**US PCs sold at the Peshàwar market:** *Computers of the US army with restricted data sold for 650$ along the road where NATO troops are attacked by the Taliban. Still full of classified information, such as names, sites, and weak points.*"

(corriere.it, 14/02/2009)

Therefore, nowadays companies think that data on mobile device must have protection: backup and encryption.

Insecurity: the deep roots

- "*Attack technology is developing in an open-source environment and is evolving rapidly*" This note makes a comparison since most of the attack tools are released open-source: it is possible to use in a simple way because there is no need to pay, and secondly if someone want to make an enhancement, he has not to rewrite the system from scratch but he can make small improvements. The way of attacking improves with small modifications in a very fast way.
- "*Defensive strategies are reactionary*": it means that we are always late. We expect a new attack, and when the attack is performed, we understand that there is a new attack, and a new protection is needed. But until that does not happen, no protections are used. The attacker is always one step ahead.
- "*Thousands – perhaps millions – of system with weak security are connected to the Internet*" We should care about the security of each computer connected to the network, not just ours. This is because everybody can be infected by malware and if it happens the computer is not more just "yours", but it is in the hands of attackers. The weakness of the security in a computer is a problem for others connected people;
- "*The explosion in use of the Internet is straining our scarce technical talent. The average level of system administrators… has decreased dramatically in the last 5 years*": with nice interfaces on computers productivity increases, but technical talent goes down (for example Linux command lines vs Linux windows interface). To be good at cybersecurity technical knowledge is required and necessary;
- "*Increasingly complex software is being written by programmers who have received no training in writing secure code*": **secure code** means code that works and cannot be easily exploited. **Security code** is some code that has got some security encryption, digital signatures, etc... Security code embeds security functions. Not all the programmers should know about security functions, but they should be aware of secure code. Every time we create an interface that require an input from user, we must check that input is in the proper form. Accept only what is acceptable and reject anything else;
- "*Attacks and attack tools transcend geography and national boundaries*": Every time we perform an investigation its always running after the attack. And it is always a problem because we start following the chain (someone attacked from Italian server, but Italian server has been attacked from French server, and so on) until we get in some countries where there is the last point with no law about cybersecurity. And if there are no law no one will know who made the attack;
- "*The difficulty of criminal investigation of cybercrime coupled with the complexity of international law means that… prosecution of computer crime is unlikely*".

Basic problems (technological)

- *The networks are insecure*:
  - Most communications are made in clear (unless you take some actions);
  - LANs operate in broadcast (sending message to everybody and "if is not for you don't read");
  - Geographical connections are NOT made through end-to-end dedicated lines but through shared lines or through third-party routers;
- Weak user authentication (normally password-based);
- There is no server authentication;
- The software contains many bugs.

Some classes of attacks

A useful means of classifying security attacks is in terms of passive attacks and active attacks. A **passive attack** attempts to learn or make use of information from the system but does not affect system resources. Examples of passive attacks are:

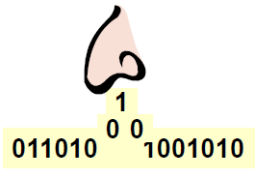
- **Packet sniffing**: the content of network packets (e.g., passwords and/or sensitive data) are read by (unauthorized) third parties;

- **Traffic analysis**: even if packets' content cannot be understood by a third part, an opponent could extract some information about the nature of the communication taken in place.

An **active attack** attempts to alter system resources or affect their operation, involving some modification of the data stream or the creation of a false stream. Examples of active attacks are:

- **IP spoofing/shadow server**: someone uses the address of another host, to take its place as a client (and hide its own actions) or as a server;
- **Connection hijacking/data spoofing**: data inserted/modified/cancelled during their transmission;
- **Denial-of-service (distributed DoS)**: the functionality of a service is limited or disrupted (e.g., ping bombing).

Packet sniffing (eavesdropping)



Packet sniffing refers to read the packets addressed to another network node. It is easy to do in a broadcast network (e.g., LAN) or at the switching nodes (e.g. router, switch). This is possible by putting the network card in promiscuous mode, which means that the card will read every packet passing through the device. This kind of attacks allows to intercept anything (password, data, …). Some possible countermeasures are: do not use broadcast networks, or encrypt the packet payload (if non-broadcast network is not possible).

Traffic analysis

It is more subtle than packet sniffing. Suppose that we had a way of masking the contents of messages or other information traffic so that opponents, so that the content of the messages cannot be understood by a third part. Even with encryption in place, the opponent could determine the location and identity of communicating hosts and could observe the frequency and length of messages being exchanged. This information might be useful in guessing the nature of the communication that was taking place.

IP spoofing (masquerading)

IP spoofing means the forging of the source network address. Typically the level 3 (IP) address is forged, but it is equally easy to forge the level 2 address (e.g. ETH, TR, ..) A better name would be **source address spoofing**. This is typically used for attacks where you do not need an answer. If everybody is in the same subnet, due to the broadcast function, it is also possible to read the replies. Attacks of this type goes for: ***data forging*** and ***unauthorized access to systems.*** The countermeasure for this type of attack is never using address-based authentication: generally, network addresses should not be trusted.

Denial-of-service (DoS)

Denial-of-service refers to keep a host busy so that it cannot provide its services. For example, in public administration for a call to a competition, offers can be sent until a date. We can make an offer and stop all the others from sending email. A possible solution is to send tons of email keeping the server busy until the message "*Message did not deliver because the destination mailbox is full*". We saturated the mail service. Other examples:

- **Mail/log saturation** as explained before;
- **Ping flooding ("ping bombing"):** the ICMP echo request usually uses a small number of bits (8 bytes) and starts a timer waiting some second for a response. We could use the biggest amount of bites possible: 64 Kbytes. We should send a lot of echo request, at maximum speed without starting timers. This will keep the host busy to answer to all these packets and will not be able to perform other tasks.
- **SYN flood:** it is a form of denial-of-service attack in which an attacker rapidly initiates a connection to a server without finalizing the connection. The server must spend resources waiting for half-opened connections, which can consume enough resources to make the system unresponsive to legitimate traffic (see TCP SYN flooding).

Usually, DoS attacks block the use of a system/device and there no countermeasures for it because it is no possible to know if someone connecting on the service is keeping busy the service on purpose or not. In other words, DoS is quite like a high increasing of customers using that service. Monitoring and oversizing can mitigate the effects. Every time there is an alert that some threshold is passed, for example resource saturation, it is time to investigate. It could be a system problem or a security problem. Security and system manager must work together.

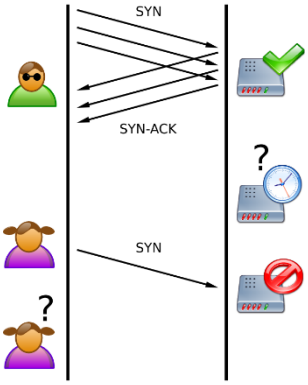


Figure 8 SYN Flood. The attacker (Mallory) sends several packets but does not send the "ACK" back to the server. The connections are hence half-opened and consuming server resources. Alice, a legitimate user, tries to connect but the server refuses to open a connection, resulting in a denial of service.

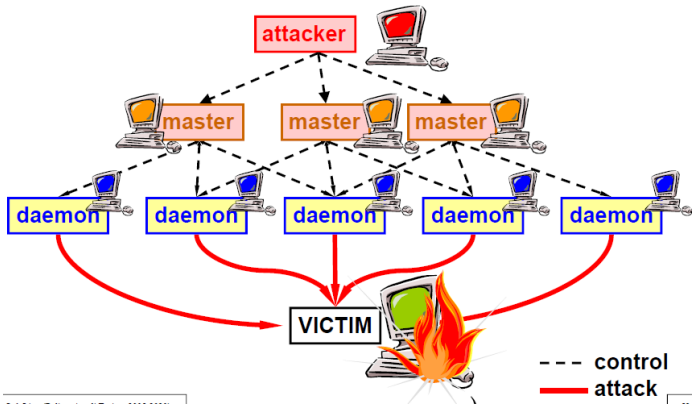
Distributed denial-of-service (DDOS)

DDOS is the same kind of attack as DOS but multiplied by the number of attackers that are attacking at the same time. Typically, a small number of individuals take controls of some nodes by installing DoS software on these, each of one is often called **daemon, zombie** or **malbot** with the purpose to create a **Botnet**. It is a network of robots (slaves) controlled by a **master** that usually uses a **C&C** (command & control) **infrastructure**, either client-server or peer-to-peer. Additionally, the communication between zombies and master is either encrypted or passes through "covert" channels (for example information goes through UDP packets contained as payload of ICMP Request messages). Covert channels are intended to mislead investigations, in fact by using UDP as payload of ICMP Request this kind of traffic can be hided, since the presence of ICMP messages is common in the normal traffic.

These bots are sophisticated: there are several cases in which they have the capability to *auto-update* themselves to the newest kind of attack and, by increasing the number of daemons, the effect of the attack can be multiplied. Other ways to improve the attack can be the use of a "**reflector",** making use of a potentially legitimate third-party component to send the attack traffic to a victim; this technique has two main advantages:

- **hiding the attackers' own identity**: the attackers send packets to the reflector servers with a source IP address set to their victim's IP (IP spoofing), therefore indirectly overwhelming the victim with the response packets;
- **multiply the effect:** can be used also an "**amplification factor"** N:1 (that depends on the protocol) where attacker typically use a *reflector server* that has a size of response that is much bigger than the size of request. For example, if you make a specific query to DNS, then the response of DNS can be as big as 70 times the request.

DDOS Attack



There is an attacker and a victim. The attacker somehow created a network of daemons, then he selects some nodes to be masters (more than one usually, for redundancy), which are part of the botnet. The attacker sends the IP address to attack, and then he *disconnects from the network* to avoid being tracked. The other masters will coordinate the work of the daemons, that must perform the attack at the same time. Masters do not directly take part to the attack, they try to remain hidden as much as possible to avoid being killed as result of an investigation, because this would stop the attack. The huge class of amplification factors that daemon can create make the victim die. Masters and daemons are not "real guys" but part of the botnet. The only human is the attacker, which disconnects after starting the DDOS attack.

Case History: DDoS towards Yahoo Server Farm

The first well known attack of this kind was in Feb 8th, 2000 at 10.30 am (PST) against Yahoo Server Farm. There was a report written by administrators. The report said:

- "*The initial flood of packets, which we later realized was in excess of 1G bits/sec, took down one of our routers …*"
- "*… after the router recovered, we lost all routing to our upstream ISP …*" this happens because an internet connection is a connection that requires 2 routers. The Yahoo people rebooted their own router, but at the other end of the communication link the router was still down.
- "*… it was somewhat difficult to tell what was going on, but at the very least we noticed lots of ICMP traffic …*"
- "*… at 1.30pm we got basic routing back up and then realized that we were under a DDoS attack*"

Later the attack was traced back to the 15-years old Canadian boy Michael Calce (aka MafiaBoy). Even if get to the attacker is not always possible, the USA lawyer can easily reach who are the daemons and get refund from them (even if daemon was a victim of the attacker, for example if the attacker can take control of a computer for DDoS purposes).

An interesting legal implication for zombie nodes: "Be Secure or Be Sued":

"There is a distinct probability that if your site has been hijacked for a denial of service attack, then you could be liable for damages. I would definitely advise clients they have grounds to sue."

Nick Lockett, e-commerce lawyer at Sidley & Austin

Case History: DDoS towards "Krebs on security" blog

On 27th Sep 2016, the Administrator of the blog received a DDoS Attacks that generated 665 Gbps and it was generated by a botnet of IoT devices (or claimed to be such). There was no use of any reflectors or amplifiers, just millions of devices that performed perfectly valid requests. It seemed like millions of users wanted to connect to the website at the same time. This generated a big traffic that even if the blog was hosted on Akamai the traffic was so big that the company had to give up and made the blog unreachable by dropping that destination on their routing table, on 29th Sep. There was unknown reason of the attack, perhaps it is connected to Krebs' analysis of similar attacks against on-line game servers.

Shadow server

In **shadow server** attacks a host manages to show itself (to victims) as a service provider without having the right to do so. There are two techniques:

- If able to **sniff request and spoof response faster than the real server**, the latter will not be able to communicate (because the 2nd package will be discarded, as it will be considered as a duplicate);
- **routing or DNS manipulation**, mapping the real name to the IP of the shadow server.

The attacks that can be carried out are:

- **Issue wrong answers**, providing thus a "wrong" service to victims instead of the real one;
- **Capture victim's data** provided to the wrong service.

The countermeasure to this type of attack is to require **server authentication**.

Connection hijacking / Man In The Middle (MITM)

It is also named **data spoofing** and this kind of attack consists to take control of a communication channel to insert, delete or manipulate the traffic. This can be achieved in a logical way (changing the route of the network) or in a physical way (if being able to reach physically a router or a switch). The attacks can be done for many reasons: reading, insertion of false data and modification of data exchanged between two parties. The

countermeasures are: **authentication**, **integrity** ("*is it the same or has been changed?*") and **serialization** (no packets added or deleted. The packets come in the same way as they were sent) of each single network packet.

Trojan / Man In The Browser (MITB)

It is called in that way because network channels are becoming more and more protected. Most of the website do not use anymore *http* but *https*: it means that performing a MITM attack is not so trivial anymore. In this type of attack, the attacker make you install some malicious software relying on users' terminals less protected (rarely smartphones are equipped with an antivirus, IoT devices have usually weak security measures, etc..) and on the ignorance of users themselves. This can be a **keylogger** recording of everything written on the keyboard or mouse clicks, or **browser extensions**. This attack is also called "Man At The End" (MATE). The security of the network connection is not a countermeasure for this type of attacks.

Zeus

Zeus attack is also known as **Zbot**. Currently it is a major malware combined with botnet. It has been discovered in 2007 and since the police is seeking for the owner of this network, the owner said that he sold it on 2010. It can be used:

- Directly: for example, MITB for keylogging or form grabbing (software that reads data in a form);
- Indirectly: to load other malware (such as the CryptoLocker ransomware).

It is difficult to discover and remove because it hides itself with stealth techniques and there are about 3.6 M of active copies only in the USA.

Software bug

Even the best software contains bugs that can be used for various aims. The easiest way to exploit software bug is to create a Denial of Service. An example was the attack against WinNT server (3.51, 4.0). The attackers discovered that WinNT server had server hosted at TCP port 135 (undocumented) and they tried to communicate to that port sending 10 random characters, and then CR. By doing this stuff the server became unavailable (100% CPU load even though no useful work is done). The problem was that at port 135 there was a new service Microsoft only that was a remote procedure call that had a bug: if received a malformed packet than it went into an infinite loop asking: "where is a good packet?". But since the ARP Server(?) is part of the kernel of the operating system it would take 100% of CPU with not possibility of being interrupted. Microsoft developed a solution with SP3 in which they corrected this bug.

Some typical application-level problems

- **Buffer overflow**
  happens when a programmer allocates a buffer of memory, but he does not verify that data being stored does not overflow that part of memory. In this way an attacker can typically inject malicious code.
- **Store sensible information in the cookies**
  in this way sensible information can be read by third parties (in transit or locally on the client)
- **Store passwords in clear in a DB**
  In this way they are readable by third parties (for example the backup operator that has the duty to make a copy of data).
- **"inventing" a protection system**
  The problem are the risks of inadequate protection

Virus & Co. (malware)

- A **virus** is a malicious program that damages the target and then duplicates himself and it is propagated by humans involuntarily.
- A **worm** damages the target indirectly, just because it replicates himself so much that it achieves resource saturation. Additionally, worm will try to propagate automatically. A worm is usually more

difficult to find because the damage created looks like normal activity unless careful analysis of the network pattern;

- The **trojan horse** is a program that is carrying some malware. It may be a valid program that also install some malware;
- The **backdoor** is a piece of software which is providing unauthorized access point (illegal but common in developers: if a developer fear to be not paid he can leave a backdoor to get access in case of payment problems);
- A **rootkit** is a set of tools that provides privileged access. It is hidden (maybe a modified program, a library, driver, kernel module or hypervisor) and stealth. For this reason, it is hard to detect and remove.

Virus and worm (malware)

Virus and worms require some kind of complicity (may be involuntary) from **the user** (gratis, free, urgent, important), **the system manager** (wrong configuration), **the producer** (automatic execution, trusted). Some countermeasures are *user awareness*, *correct configuration/secure software*, *antivirus* (installed and updated).

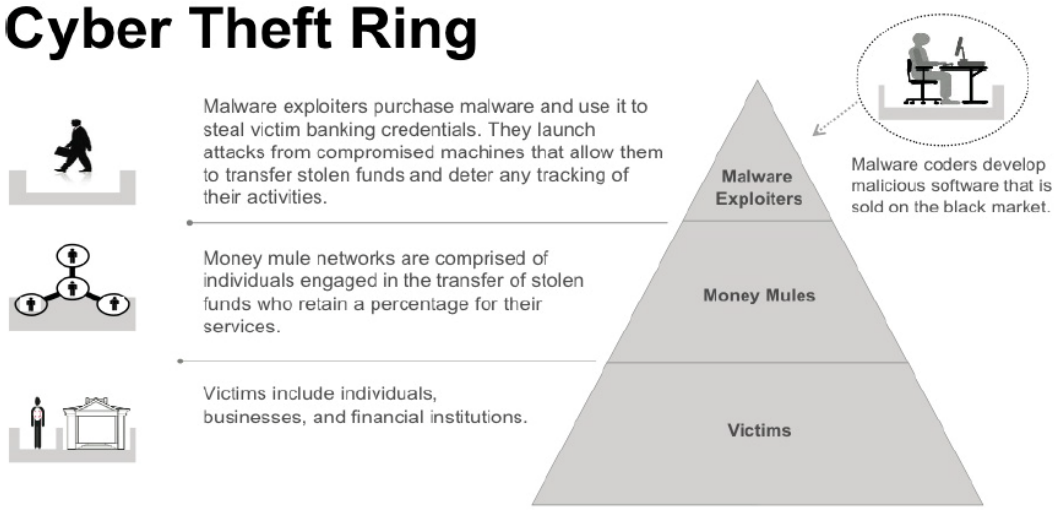
Malware food chain

The vulnerability discovered by someone can be used for developing some malicious code (for example to get access to a page and change some data). This information can be exploited in two ways:

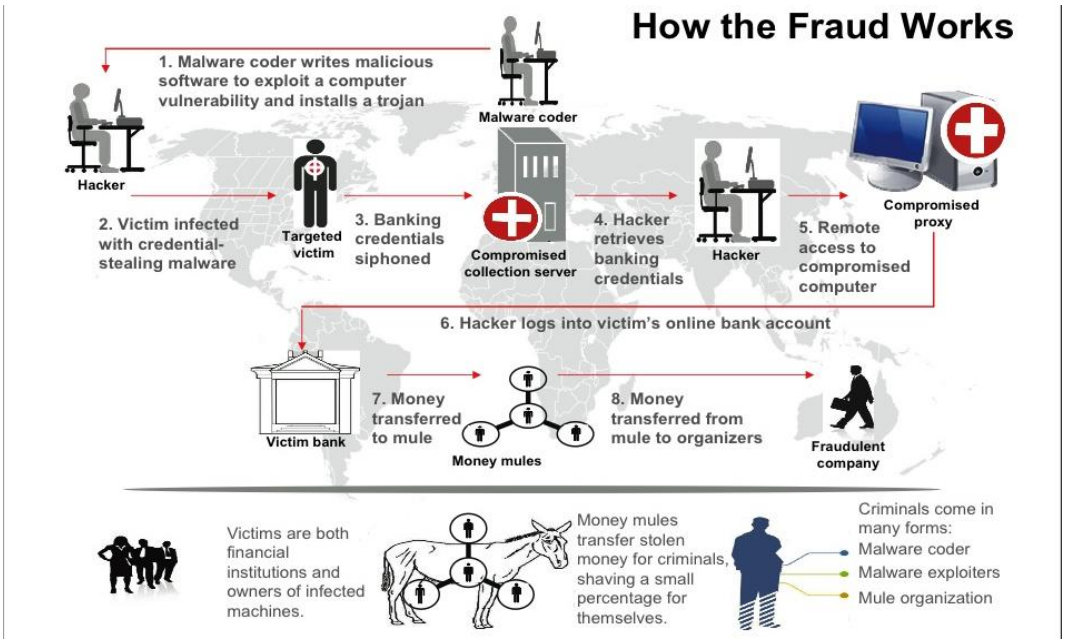
- Perform the attack just to be in a sort of "Hall of Fame", especially in the past;
- Most recently, for business, sell the information about the vulnerability or a working code that exploits it on the **vulnerability marketplace**, which is a hidden marketplace that takes place only on internet on some dummy servers that appear for a couple of hours at night.

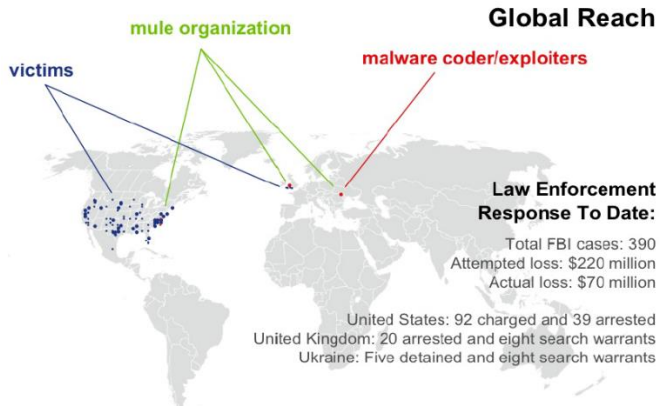
You can pay with bitcoins to buy/sell the vulnerability. The people who buy those vulnerability are the **malware toolkit makers**, people who create attack programs that needs malware in it. This kind of software is then used by malware distributors (spammers, web owners).

Zeus



It often happens to receive an e-mail which ask to send money outside of the country in exchange of a lot amount of money. If you accept this request, you will get the money. They will start trying to make a test: sending a small amount of money (for example $10.000) to you and asking you to send $7.000 to Nigeria, and to keep the $3.000 for you. If you agree you will get the money because they are stealing money from someone's bank account and send them to you (so you appear to be the responsible of that) and you will send them via MoneyGram. The people in Nigeria will run away, but you will be the responsible for police. This is a common case of **money mule** (accepting money from a stolen bank account and sending them to the destination).





Most of the victims are based in US or UK. The mule organization is spread across many countries: US, Europe, British Ireland, but malware coders/exploiters are typically in the East Europe. In the picture there are also numbers about the Law Enforcement Response.

Ransomware

A ransomware is a malware oriented to get a **ransom** (asking money to release something). If ransomware afflicts a computer, it typically makes the content of the disk unreadable (typically encrypted). It is valid also for tablet and smartphone (for example changing password for accessing data). Not always, after you pay the required amount of money (typically in bitcoin), your data can be recovered: this happens because the key to unlock files is stored typically on a server that could be closed due to investigation from police. In this way the attacker has no more the ownership of the key used for locking data.

Ransomware-as-a-service

For some time, there was a ransomware-as-a-service, in which someone provided the source of ransomware to use it even if you do not understand anything of it. It was called **TOX malware (server in the TOR anonymous network)** which ask for the ransom and handles the payment (with a 20% service fee). The "customer" has the only task to distribute it to the victims (for example by leaving a USB pen in public places). It had a fast growth (1000 customers/week, 100 infections/hour).

Even if TOX was stopped from police, ransomware still exist. Technology is not enough to protect against ransomware. Surely it helps to have a good anti-malware system installed. This kind of attack is not just for end user, but it goes also for servers.

- **Encrypted data**

Ransomware usually encrypt data that can only be restored with a proper key. **Backup** can be a possible solution.

- **How old is the backup?**
  Backup could be old but let us assume we make a backup every day. Ransomware developers know about the backups, so they developed the **silent ransomware**. The ransomware is installed in the system, but it does not encrypt your disk, but it encrypts your backups. It continues to do that for many days (making backups unreadable) and only later the backups are discovered to be corrupted;
- **Off-line or network backup?**
  Backups should be made offline. Just consider a real case of a dentist studio where a Network-Attached-Storage (NAS) was used to perform continuous backup of the local storage on the network storage: a ransomware found this type of configuration and also encrypted the data on the network storage, making useless the backups.
  - But, even if performing backup offline (connecting the external hard disk only during backup and then detach it), is it possible that a silent ransomware can see the external storage when connected and encrypt it?
    Yes, that is possible. To perform a good backup an **inverted backup** technique should be used: the backup disk is not attached directly to the target pc (the one whose data are to be backed-up), but there should be another node that remotely mounts the disk of the target PC, that it is seen just as a device, makes a copy and then disconnect from the target PC. In this way, even if the target PC is infected, this will not affect the backup process. Remark that this node that performs the backup must NOT be connected to the network.
    It is anyway possible that the malware has affected the driver used by the target PC to export the disk (i.e. the NFS daemon in Linux), but it is unlikely.

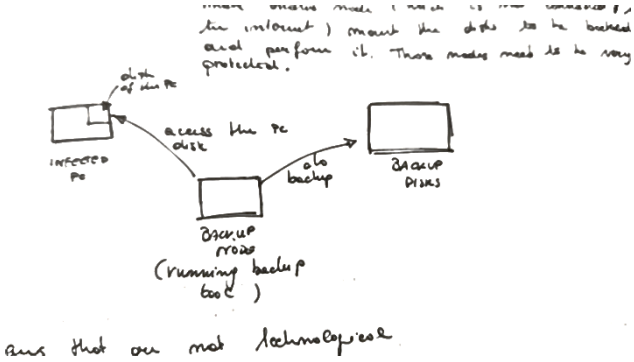


Figure 9 A scheme that should resemble the mechanism of an inverted backup

- **Verified or "trusted" backup?**
  The question is: did you verified the correctness of the backup or simply trust it? There is a famous real case in Sweden of backups consisting in magnetic tapes being corrupted because of during the transportation the heated seat generated current on wires, generating magnetic fields that damaged the tapes. This fact was discovered only when data were corrupted and there was the need to look to backups.
- **When was the attack?**
  To know which is the correct backup, it is necessary to know when the attack happened. There was the case of a video archive where a journalist asked to consult the video archive. The journalist looking for videoclips noticed that sometime the name of the files was right, other times it was not right. Yes, there is a backup, but which backup is the correct one? When was the attack done? If you do not know when the attack happened, it is not possible to know which backup is correct to restore.

Basic problems (non-technological)

The picture tells a big truth: it depicts a match between all the technologies on the left such as firewall, antivirus (data security) while on the right there is the human error, that is the problem. The purpose of technology can be defeated by a human error (intentional or not).

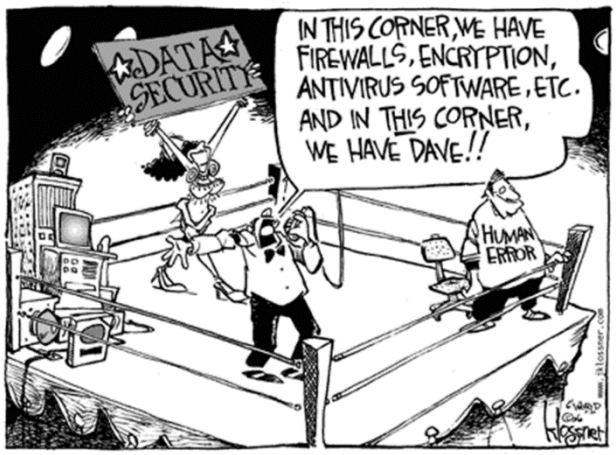


Figure 10 Even the most sophisticated technological tools can be made useless by human error

- **Low problem understanding (awareness)** People in general do not understand that there is a security problem. Furthermore, people do not have security measures and they install them only after the problem has come. This happens because they do not think a problem will happen to them.
- **Mistakes of human beings (especially when overloaded, stressed):** A lot more frequent when we are stressed or there is a lot of work.
- **Human beings have a natural tendency to trust** We usually do not suspect about other people. Since online we do not meet people, it is easy to have tendency to trust other people.
- **Complex interfaces/architectures can mislead the user and originate erroneous behaviors**
- **Performance decrease due to the application of security measures**
  Example: a company asked Lioy to support them for antivirus. This company claimed to have the best antivirus, but they were frequently infected with viruses that blocked their system. Lioy went to that company's building and considered everything (antivirus installed and updated). Everything seemed to be all right. Lioy started walking through the office to understand the problem: walking through he noticed a desktop with the icon of the antivirus disabled. Lioy asked to the employee why it was offline, and the employee answered that the antivirus checks every file is read/write and this increases the waiting file to get a file. In this way the employee found a way to disable the antivirus, work with the computer, and then open it again. All the employees make it in this way in the office. Lioy went to the head office and told them they had a non-technological problem.

Social engineering

In the context of information security, *social engineering is the psychological manipulation of people into performing actions or divulging confidential information*. Usually the target are **naïve users** (for example "change immediately your password with the following one, because your PC Is under attack") but also **experienced users** are targeted too (for example by copying an authentic mail but changing its attachment or URL) and it happens via mail, phone or even paper.

Examples of Social Engineering

- **Phishing** (pronounced "fishing"):
  It is an attack technique based on attracting a network service user to a fake server (shadow server), using mail or IM, for acquiring his authentication credentials or other personal information, or persuading him to install a plugin or extension which turns to be a virus or a Trojan. An example message can look like: "*Dear Internet banking user, please fill in the attached module and return it to us ASAP according to the privacy law 675…*". There are some variants:
  - **Spear phishing**, when the message includes several personal data to disguise the fake message as a good one (e.g., mail address, name of Dept/Office, phone no., etc.);
  - **Whaling,** when the target user holds a position of responsibility or power, such as a CEO or CIO (see below). This has two main advantages from the attacker's point of view: the damage done by deceiving these types of people can be much greater, as their credentials can give the attacker great power; these people generally occupy a commercial and administrative position, which usually means that they do not have the necessary knowledge to handle security issues;
- **Psychological pressure**: usually this is achieved in two ways:
  - By exploiting human empathy towards a friend or a co-worker, for example generating a fake message conveying a message like "Help me, otherwise I'll be in troubles …", leading to do

some action that could be against the best practices, hence creating an opportunity for an attack;
    - By pretending to be a boss, also asking something that could forbidden, under the threat "do it, or I'll report it to your boss …" (see below).

  In general**,** these people who run this kind of psychological attacks try showing acquaintance with the company's procedures, habits, and personnel, for gaining trust and make the target lower his **defenses**.
- **Pharming**: it is a term of controversial use, it embodies a set of several techniques to re-direct a user towards a shadow server, by:
    - changing the "hosts" file at the client;
    - changing the nameserver pointers at the client;
    - changing the nameservers at a DHCP server (e.g., an ADSL / wireless router);
    - poisoning the cache of a nameserver.

  This could happen via **direct attack** if there is a vulnerability or misconfiguration, or via **indirect attack** with a virus or worm

Fake mail/IM

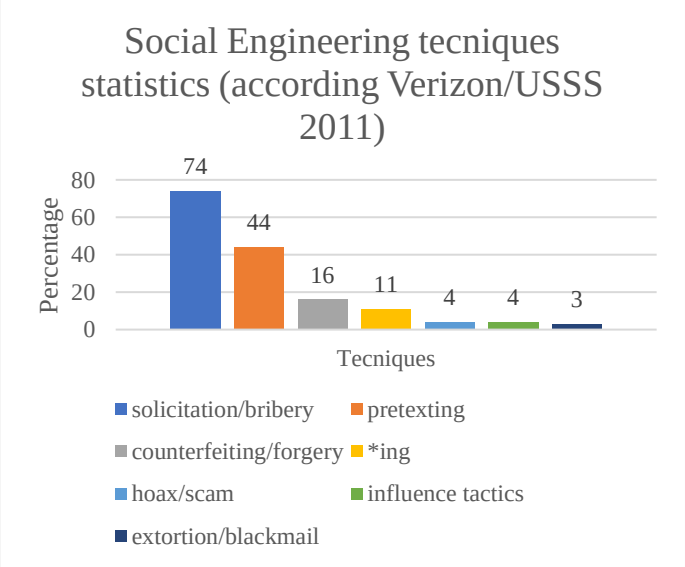
It is easy to create a fake mail, but it is difficult to use the right "tone" (for example "*do I sign the emails as Antonio Lioy, or just Antonio, or A. L.?*") otherwise the fake mail will be detected. It is also possible to create a fake SMS or IM. For example, can be sent:

- *Fake ATM withdrawal messages* asking you to call a number that will ask your credentials.
- *Fake kidnapping alarm*: this happens often typically by IM. It often happens that many old men go to the police saying, "help me because my girl has been kidnapped and they want 200.000 euro to release her". This girl is a woman met on socials that creates a fake relationship with the man to create empathy and make the victim give money to the fake girl.

Case History: "Mr. Confindustria in Brussels tricked by a hacker: 500.000 Euro lost. Fired." (Repubblica journal on 30th September 2017)

"Transfer immediately half a million to this foreign bank account. But this mail was from a hacker. And the money has disappeared. The fake order was (apparently) signed by director Panucci: "Please execute and don't call me because I'm out of office with the president." …".

Final message: all the employees should be trained in the security problems of nowadays life.



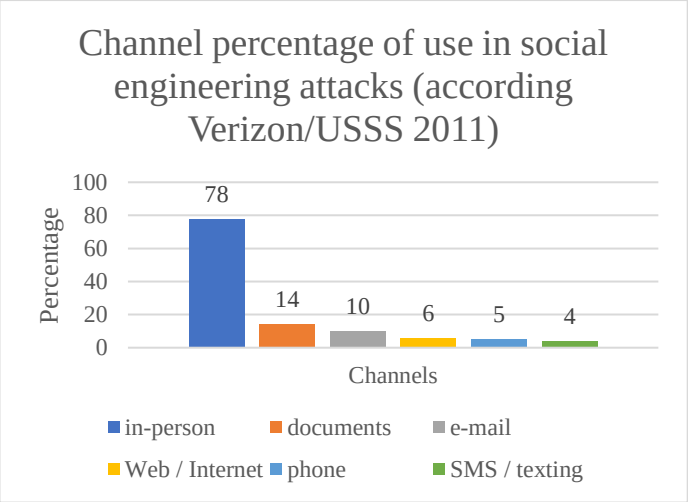
Social engineering techniques

Statistics from an old survey of Verizon where the important point is that when trying to explain social engineering to employee, they must be aware of each kind of attack, as following:

- (74%) solicitation / bribery;
- (44%) pretexting;
- (16%) counterfeiting / forgery;
- (11%) *ing (i.e. phishing, pharming, ...);
- (4%) hoax / scam;
- (4%) influence tactics;
- (3%) extortion / blackmail.

Social engineering channels

The same happens for the channels that are being used. It could be quite surprising but, in this statistic, the 78% of the attacks of social engineering were in-person, which means that someone showed up pretending to be someone else.



Now we will consider some famous attacks to understand what made them possible: when there is an attack the focus should not be on the attacker, but on what is it possible to learn from the attack.

Case History: T.J. Maxx attack (2007)

In 2007 there was an attack against T.J. Maxx (a chain of shops). The attack happened from the external where it was possible for the attacker to access 45 million of credit cards numbers from customers. The attack was a long lasting one, about 18 months (up to January 2007). It became famous because a group of 300 banks damaged from the attack (because they had to refund customers) intended a class action against T.J. Maxx, asking for a refund of 10 million dollars. This happened because, even if in 2007 it was known that the wireless protocol WEP was insecure, T.J. Maxx kept using WEP rather than WPA. The attacker did not need to enter in the shop, instead he was sitting in the park with a laptop and proper software, intercepting all the transactions between the cash register and the back-end server. The attack was performed by 10 people with the help of an ex-cracker, then hired by the US secret service: this is not strange, in fact when an attacker is found, instead of going to jail, often he can work for the government as a consultant or by stopping the attacks.

Considering the legislation, in general the law does not prescribe what specific security measure to be taken, but just warns to set up defences according to the state of art. That means that a manager of security must always be on the edge to apply the most recent solutions against new kind of attacks.

Case History: US Air Force phishing test transforms into a problem (04/2010)

In April 2010, during an Operational Readiness Exercise (ORE) of the Andersen Air Force Base located in the Guam island, the reaction of airmen to a phishing e-mail has been tested. The e-mail sent by security testers said that crews were going to start filming "Transformers 3" on Guam and invited airmen to fill out applications on a Web site if they wanted to work the shoot. The Web site then asked them for sensitive information, and the airmen should have been trained not to provide that kind of information. The outcome of this exercise became viral because one of the airmen posted this news outside the base, reaching the civilian world. As the rumour spread that the hotly anticipated film was coming to Guam, local media started calling the base, which then began the work of setting the record straight.

"Leadership from Andersen AFB regrets that there has been any confusion in the general public regarding this exercise phishing attempt," Andersen said in a statement. "We hope however that this will show that all individuals need to be careful about the real danger of phishing emails and that others can learn from this exercise."

Some important recent attacks

The first attack discussed is **Stuxnet**, the first cyber-attack that created physical damage. Then **Black Energy**, the first of most well-known attacks used against a critical infrastructure (for example the electric greed distribution). And finally, **Mirai, BlueBorne** and **BrickerBot** which are all attacks for IoT, embedded systems, automotive sector, and home devices.

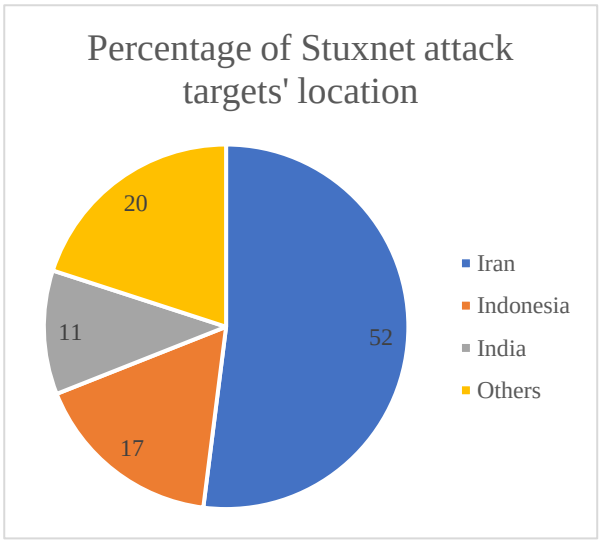
Against cyber-physical systems

Stuxnet (2010)

It is important because it is the prototype of a new kind of attack, since it caused physical damage (like a virus) and it attempts to propagate itself to other systems (like a worm): the target were **SCADA systems** of a specific manufacturer attached to the infected nodes. SCADA stands for Supervisory Control and Data Acquisition, and they are those some computer systems that control process plants or machinery, for example production system for cars, food and so on. Since SCADA is the interface between computer and physical reality, an attack against it could damage physical system components. Stuxnet was a very sophisticated attacked because this worm contained inside four attack vectors:

- one was based on an already known vulnerability (for which a patch existed);
- one exploited already known vulnerability, but for which no patch was available (for those who got the patch for the first one);
- two "zero-day" vulnerabilities (if there were countermeasures for the known vulnerability)

Attack motivations



Even the timing and the location is peculiar. First, on 17/6/10 the first damage was created and knew that Stuxnet was existing. For this reason, investigation started and one week later 24/6/10 the analysers detected the use of a first signature certificate and revoked it on 17/7/10, but then the malware started using a second signature certificate. Since the discovery, it took one month for security bulletins by CERT (Computer Emergency Response Team) and MS (Microsoft). Then, Microsoft started the develop of patches that gradually released through October '10. This malware self-stopped its propagation on 24/6/2012 (that was written inside the code of itself): this suggests that this malware was targeted to a specific time window.

Most of the attacks are against the most developed countries, but this malware was for 52% in Iran (the geographic distribution is depicted in the following graph). It seems to be targeted to a specific area: t*his worm had the target to destroy the uranium enrichment plant in Iran*.

Attack mode

It seems that the first infection came through a USB pen inserted in one of the computers that managed the SCADA system. Then it propagated through all the computers connected to the network of the enrichment plant in Iran thanks to shared disks, Microsoft Spooler Subsystem and RPC services bugs. The infection via USB key was possible because inside that network of computer there was no internet connection, and a USB Key was required for updates of SCADA systems. It seems that the technician arrived from another country for the maintenance with the USB pen (and good software in it), got distracted in the hotel from some girls while someone added to the USB pen the malware. This malware was disguised as a driver (pretended to be a driver) with a digital signature validated by Microsoft and used two different certificates. Additionally, the guilty was not only from Microsoft but also of the developers, because the SCADA system had only one shared default password for the back-end database.

Attack consequences

The malware was able to push the spin of cylinders (used to enrich the uranium) over the limit, without alerting the monitor system used by technicians, who managed to stop the system when the damage was already done. This delayed Iran Uranium Program for several years, to restore correct operation of the implants.

Lessons learnt from this attack:

- Physical system separation (air gap) does not imply security: in fact, without any other standard protections (anti-virus, OS patches, firewall) these systems are not secure. In this specific case, the systems did not use any kind of standard protection, relying too much on the physical barrier;
- Unnecessary active services can be source of vulnerabilities: MS-RPC, shared network print queues, shared network disks were not necessary but still running. In general, any unnecessary service should be disabled;
- Validation list for software to be installed should be used: it seems that the technician copied all the content of his USB pen, also copying the added malware. He should have copied only the file necessary to the SCADA systems update.

Stuxnet's brothers

Stuxnet's brothers are named in this way because they have the same development platform as Stuxnet. The development platform is called "*tilded"* because by performing reverse engineering of the malware has been discovered that all the variables are named "tilde1", "tilde2" and so on…

Duqu has been discovered in September 2011 and it is not a worm nor a virus (it does not provide any damage and it do not propagate itself). Once Duqu is installed it gathers and send system info for attack preparation (reconnaissance & intelligence, which means understanding the kind of system where it is installed and to understand what the feature of the system are, such as the software installed).

Flame has been discovered in May 2012 and it is a *system spyware* which records network traffic, audio, video, and keyboard (silently). Again, it has no physical damages and it spreads via USB or network. The purpose of a spyware is to stay inside a system as long as possible in order to spy. It has also a *backdoor* to make possible the remote configuration and the update of the spyware. Flame was active for two years before its recognition.

Sauron – one malware to rule them all

Sauron has been disclosed in August 2016, but it was active since 2011. It claims to be created by the Strider group (anonymous group of attackers). It has a **remsec malware**, which is a stealthy backdoor and a logger. It was very selective for his targets (some individuals in Russia, an airline in China, an organization in Sweden and an embassy in Belgium): only 36 infections since 2011. Remsec malware is actually a "**loader**", while the attack is written using **LUA module**, that is a new interpreted language. It is just needed to write the code in LUA and then the loader runs it independent from the type of system. LUA modules are: net loader (load from the net), host loader (load from USB pen), keylogger (logging keyboard use), net listener (recording packets sent/received), basic/advanced pipe (put something after another before sending something to the network card, the pipe is in the middle to alter or read data) and HTTP back-door.

Normally it is installed in a computer connected to the network, but it is very flexible because it can also collect data from air-gapped computers, export them in a USB key and send them to the internet as the pen is used on a connected computer. Difference in size of the contents of the USB key can be hidden by copying the data and then marking the corresponding location inside the device as "broken" memory unit (e.g., page), so that the OS will not try to read or write those locations, but the attacker nevertheless knows that their content is the information stolen from the victim.

Against critical infrastructure

Black Energy

Black energy belongs to the category of "**Advanced Persistent Threat**" (**APT**) which is the fact the system is compromised with some sophisticated techniques and it is continuously monitored from an external command and control system, typically by very motivated people (i.e., the threat) to remain undetected as long as possible (that's why it is named Persistent).

Black Energy is the third evolution step (more evolutions are unknown): DDoS attack in 2007, attack against Industrial Control System in 2010 and then SCADA in 2014. It propagates through the KillDisk ransomware.

Now BE is at version 4. Black Energy has become famous and has got this name because it was used to attack the Ukrainian Energy Network. For up to 6h the electricity of 230k Ukrainian customers was down.

Basically, Black Energy was a *Trojan* which spread with spear-phishing emails or malicious Microsoft Office documents (VBE extensions). BE consists of several different components with customized functionality by perpetrators on the victims:

- Access and modify files;
- Bypass access control mechanisms and collect credentials;
- Collect data on the compromised system;
- Use the compromised system for network vulnerability scanning;
- Spy victims, from screenshots to microphone;
- Runs apps and starts services.

Against Internet-of-Things, automotive, home

Mirai

Mirai is the most famous IOT **cyberworm** discovered around September-November 2016. This kind of attack does not damage directly but simply saturates the resources (like the DoS). Every system infected by Mirai become part of a huge botnet for a large-scale DDoS. It was used for disruptive attacks such as:

- Krebs on Security → up to 620 Gbit/s
- Ars Technica → up to 1Tbit/s
- Dyn DNS provider → requests from tens of millions of IPs

Botnets are deployed into millions of IoT devices such as cameras, residential routers or baby monitors, because in these devices there is almost no protection installed. In this way, Mirai can easily spreads into home networks. The fast networks (such as 4G and 5G) make it even more critical because these devices can send a lot of data per second. Mirai is a very complex malware because it has almost no external dependencies (this means that the malware is compiled with static libraries) and it is cross compiled to be executed on several platforms. Mirai is released as an open-source worm: this means that not only it is possible to study it, but also that new variants can be easily developed from other attackers. Mirai contains a propagation scan phase to compromise additional targets (starts contacting other nodes if it can infect them). It also observes the victim system before contacting the C&C: when Mirai is installed on a device, it doesn't know if it is a true or fake device (maybe used to study Mirai's behaviour) and for this reasons when it starts, it connects to a fake C&C and if the communication works it means that there is a problem (because connection should not work, which means that someone is giving to Mirai full access) and it shouts down automatically. Only after performing this check, it contacts the real C&C. The kind of attacks, since is DoS network based, is driven by C&C that can decide which of attack to make and it can open many TCP connections or send tons of UDP packets or using GRE or simple SYN flooding.

Mirai code was disclosed weeks before the first DDoS attack to make impossible to recognize the perpetrators. Since then, there are lots of clones for example:

- 900.000 Deutsche Telekom routers were compromised. They were routers manufactured with Arcadian which had a well-known bug TR-064 and yet DT did not fix them: so Mirai was easily installed. Mirai can attack also other routers' vulnerabilities such as TalkTalk protocol (when enabled).

BashLite

First famous *malware* for IoT, mainly targeting cameras/DVRs. In this case those elements had an embedded Linux system, and the attack exploited the Shellshock bug in the Bash shell. Some attacks where orders of magnitude less effective than Mirai (some attacks reached the peak of 400 Mbit/s) and it immediately contacts C&C and then propagates.

BlueBorne

BlueBorne uses Bluetooth connections to control target devices: it exploits several known vulnerabilities for example buffer overflow in memory management of Linux/Android network stack. It also bypasses security controls usually placed at the "expected" border and it affects air gapped systems, IoT and automotive devices: usually developers of automotive systems place defences, but only at the "expected border", not considering that Bluetooth could be an attack channel.

As in general for attacks, if one fails to recognize all the attack channels to a system, there is the risk to protect some and forget about others. BlueBorne also spread through the air which means faster diffusion. It can perform Man in the Middle or take full control of the victim.

Immuni example: now that all people have the app installed, which requires Bluetooth, people might try to get the access to the device through Bluetooth.

This requires the design of new protections, which are not easily done. Some firms claim they have a new program that provide a mitigation to block execution of unwanted code (detected and injected through Bluetooth).

BrickerBot

The Wind-Infostrada case (fiber subscribers)

A message appeared on blogs and chat (message on the left side). In this case Wind had to physically replace all the modems because they were not able to fix it from remote. It happened on October 2017. Modems got out of order, which means that they had to be replaced. The author of this attack claimed he did that for the good, maybe to avoid the creation of a botnet. This case was similar, but worse, to the Deutsche Telekom case in 2016.

Lessons learnt:

- **Use strong passwords**;
- **Permit external administrative access only from specific "trusted" networks (and not from any address on internet)**: using IP addresses as "filter" to allow access is anyway weak, but at least is better than allowing it from all;
- **Apply timely all security patches to minimise the WOE**: in the Wind-Infostrada case, the bug was well-known, so the company should have installed the patches.

Real-time attack maps

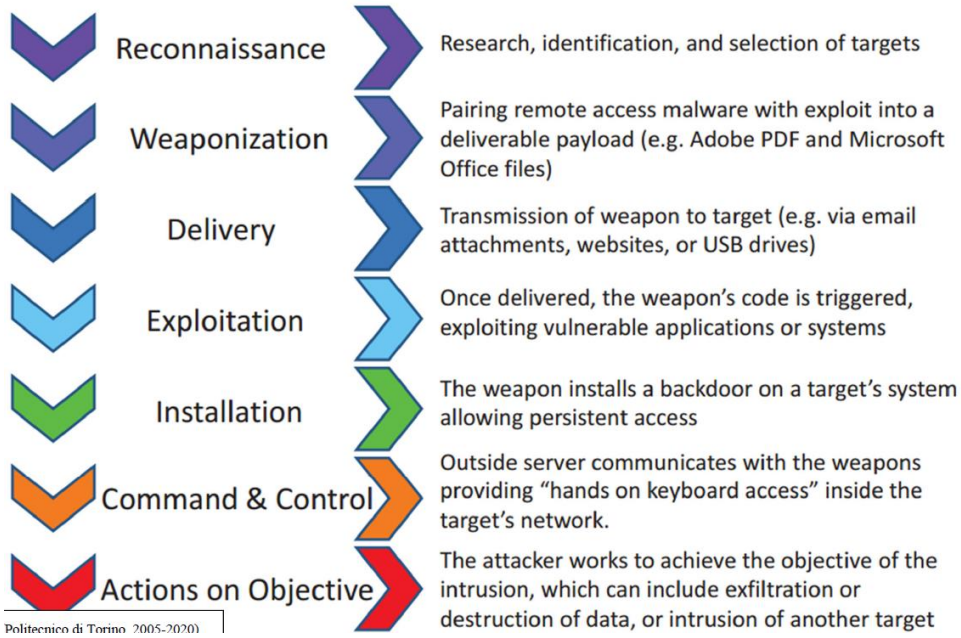
There are many websites that can show how many attacks are being performed. It is possible because many of these sites are providers of security solutions. They get the information from their own software which is installed on millions of devices and detects possibly attacks. The websites have many labels which tell exactly what attack is:

- ***OAS*** = *On-Access Scan*
  Anti-malware that perform a scan to detect if there is a malware before accessing a file.
- ***ODS*** = *On-Demand Scan*
  Scan on inserted USB pen before files are copied.
- ***MAV*** = *Mail Anti-Virus*
  Antivirus on mails that will check for the attachment.

- ***WAV*** = *Web Anti-Virus*
  Before the download of something from internet, the item is temporarily stored in a central facility, checked for possible malwares and only if there is nothing it will be transmitted to the destination.
- ***IDS*** = *Intrusion Detection System*
  Monitors networks and hosts for possible attacks.
- ***VUL*** = *VULnerability scan*
  More pro-active: some machines periodically try to contact others on network and check for a vulnerability: if something is found, a warn is sent.
- ***KAS*** = *Kaspersky Anti-Spam*
  Specifically for Kaspersky
- ***BAD*** = *Botnet Activity Detection*
  Botnet detected performing activity.

Cyber (intrusion) kill chain: attack

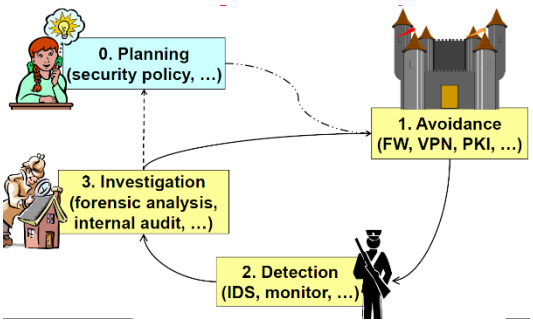
Based on the approach taken, in some cases this following table can be useful to describe an attack.



This is a possible sequence of steps of an attack. Someone says that it is not perfect because for example the phase 2 may include "development" phase (e.g., buy useful domains, tools, servers). It has also critics because this description is based on "perimeter defensive model", but what about insiders? It may not work for that. There are also several phases that are performed externally to the target and thus they are difficult to identify (cyber intelligence!). Countermeasures exist for nearly every phase:

- ***Detect***: determine whether an attacker is poking around;
- ***Deny***: prevent information disclosure and unauthorized access;
- ***Disrupt***: stop or change outbound traffic (to attacker);
- ***Degrade***: counter-attack command and control (by sending data to that C&C);
- ***Deceive***: interfere with command and control (change traffic to the C&C);
- ***Contain***: network segmentation changes (isolate the intrusion).

The three pillars of security



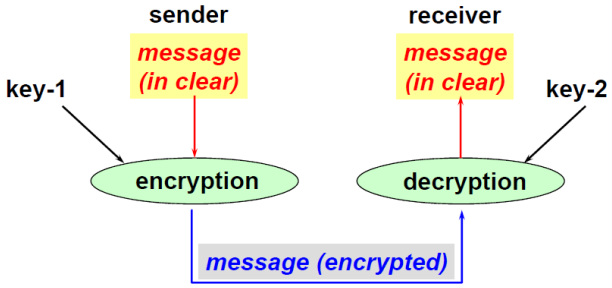
- **Planning:** define a security policy to implement;
- **Avoidance:** defences such as firewall, VPN, PKI

If defenses are overstepped:

- **Detection**: intrusion detection system or monitoring;
- **Investigation**: once attacked perform forensic analysis, internal audit (understand what went wrong);

The last step must be done **before** adjusting the system.

Basics of ICT security



The most used technique to achieve protection since many centuries is **cryptography**: a mathematical technique which perform transformations consisting of the algorithms of **encryption** and **decryption**:

- the encryption algorithm takes a message (in clear) and transform it in such way which is no more understandable;
- to recover the original text, the inverse algorithm of decryption will make it readable again.

Next to the algorithms it is needed also a **key**, key-1 for encryption and key-2 for decryption, which are just a stream of bits.

Cryptography is used in communication and to store data (for example to store data on disks without the permission to read them except us). The common terminology used in cryptography includes other two keywords:

- **Plaintext** or **cleartext**: the message in clear, it will normally be referred using **P;**
- **Ciphertext**: the encrypted messages, it will normally be referred using **C.** In some countries "encrypted" sounds offensive for religious reasons (cult of dead); in those cases, "*enciphered*" is preferred.

Cryptography's strength (Kerchoffs' principle)

Kerckhoffs' Principle (1883) states that the security of a cryptosystem must lie in the choice of its keys only; everything else (including the algorithm itself) should be considered of public knowledge. However, this principle relies on the fact that the keys have the following properties:

- Are kept **secret**;
- Are managed only by **trusted systems**;
- Are of **adequate length.**

Not only it has no importance that the encryption and decryption algorithms are kept secret, but it is better to make the algorithms public so that they can be widely analysed, and their possible flaws and vulnerabilities identified.

Security through obscurity (STO)

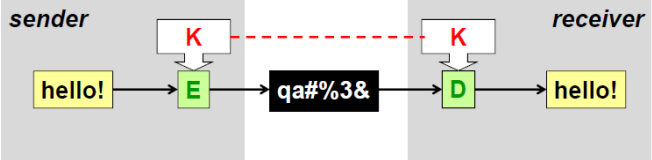
The Kerckhoffs' Principle is related to the concept of **Security through obscurity:** it means that a system is protected but the details on how it has been protected are not disclosed. Generally, this alone is not considered a valid security mechanism, because if someone discovers how the systems has been protected (and we saw that there are also non-technical ways by which this can be achieved), it is no more secured. For this reason,

we say that "***Security through obscurity is a thing as bad with computer systems as it is with women***": "*Men try to hide things to women, but when they discover the truth, it is worse than if they discovered it from the beginning*" [Cit. Lioy].

However, there is a category of people (such as military men) which tend to apply STO, but as an additional layer. It is possible to use STO as a layer only if a really strong algorithm is used (but not a secret one).

Depending on which relation exists between key-1 and key-2 there are different kinds of cryptography.

Secret key / symmetric cryptography



It has this name because is used only a single key shared by sender and receiver **only**. In the picture there is a plaintext which is used as input for the E (encryption) block also inserting the key. The result is an understandable text that is sent to the receiver. To recover the text the D (decryption) block algorithm is used with the **same** key used to encrypt the original text. If a different key is used, an output is available, but it will be wrong (and typically understandable). The problem in the picture is the dashed line: how to share securely the key among sender and receiver? The formulas used are the following:

$$C = enc(K, P) \text{ or } C = \{P\}\,K \qquad\qquad P = dec(K, C) = enc^{-1}(K, C)$$

The key is normally in the first position because it permits to use P or C by having the key in evidence. This kind of cryptography has a low computational load, and it is used for data encryption.

Symmetric algorithms

name	*key*	*block*	*note*
DES	56 bit	64 bit	obsolete
3-DES	112 bit	64 bit	56-112 bit strength
3-DES	168 bit	64 bit	112 bit strength
IDEA	128 bit	64 bit	
RC2	8-1024 bit	64 bit	usually K=64 bit
RC4	variable	stream	secret
RC5	0-2048 bit	1-256 bit	optimal when B=2W
AES	128-256 bit	128 bit	most common

Figure 11 The table shows two important information about some of well-known algorithms: the length of key used and the basic unit of data which the algorithm is able to handle. See down below that there are methods used when data size is not equal to one block.

There are many algorithms and the table on the left is just a small selection. The first column gives the name, the second gives the key length and the third gives the basic unit each algorithm can encrypt: **block algorithms** need a minimum unit of data. From the list, only RC4 is a **stream algorithm** that does not require a fixed block, but it works on any length. It is also a secret algorithm and there is no information about it (no specification or formal proof that it is really secure). The DES algorithm (for many years the standard) are now considered obsolete and should never be used. The most common/used algorithm of this kind is **AES**, currently the strongest. **RC5** is optimal when the block size is the double of the word of the CPU architecture (i.e., 64-bit arch -> 128-bit block) on which the algorithm is implemented.

Why there are so many algorithms? Because we have many kinds of computers and many algorithms are not suitable with low CPU power.

The EX-OR (XOR) function

⊕	0	1
0	0	1
1	1	0

It is the ideal "confusion" operator. The peculiarity of this truth table is that it has the 50% of 0 and the 50% of 1. If XOR is performed with 2 random inputs (probability 0:1 = 50%:50%) then also the output will be equally random. (for example AND has 0 more likely). XOR does not change the probability distribution of the input although it generates different outputs.

DES

It stands for "**Data Encryption Standard**" and it is a standard *FIPS 46/2* (**F**ederal **I**nformation **P**rocessing **S**tandard, the body which standards solutions for the American government). The mode to apply DES to data which is not equally divided in blocks are mentioned in the standard *FIPS 81*. DES is a strange algorithm because it has got a 64-bits key, but its effectiveness is only then one of a 56-bit key, since 8 bits are the parity of the others. This means that when a key is built for DES algorithm only 56-bits are created and every 7 bits the algorithm puts a bit which is the parity of the previous 7 bits. When an attacker needs to attack DES, is needed to discover only 56-bits. This is the only algorithm that has a difference between the **actual** (bits used to create key) and the **effective** (total bits amount) bits. DES uses a block which has 64-bits data block designed in 1960s when computer where not so powerful. To perform all the mathematical computation, it was needed to create a special purpose unit called **encryption processor** because the DES uses:

- **XOR**: which is not a problem → elementary operation;
- **Shift:** not a problem → elementary operation;
- **Permutation:** expensive operation. The permutation is not random, there are several ones, but still implementing that was much more efficient if done directly in hardware.

Triple DES (3DES, TDES)

It is the repeated application of DES (three times). It is possible to use two or three different 56 bits keys. It is usually applied by taking the input, encrypting it, and repeating the process other two times using as input the result of the previous encryption operation (so that three encryptions are applied in a row). Normally, it is implemented in **EDE** mode, which in its standard form requires two keys: the plaintext is encrypted with key-1, then on the output it is applied the decryption algorithm with key-2 (that actually does not decrypt but apply anyway a transformation) and finally this last output is again encrypted using key-1. This mode was selected because by putting $K1 = K2 = K3$ with the EDE is implemented also the simple DES. So EEE works, but EDE works equally well. It has been shown that 3DES with two keys can have lower security guarantees than expected; in particular, naming K_{eq} the equivalent key obtained by applying the discussed transformations:

- **3DES with 2 keys has** $K_{eq} = 56\ bit\ if\ 2^{59}B$ of memory is available by the attacker, otherwise $K_{eq} = 112\ bit$. The transformations applied are summarized by:
$$\boldsymbol{C' = enc(K_1, P)} \qquad \boldsymbol{C'' = dec(K_2.C')} \qquad \boldsymbol{C = enc(K_1, C'')}$$
- **3DES with 3 keys** ensures $K_{eq} = 112\ bit$:
$$\boldsymbol{C' = enc(K_1, P)} \qquad \boldsymbol{C'' = dec(K_2.C')} \qquad \boldsymbol{C = enc(K_3, C'')}$$

The 3DES is a standard FIPS 46/3 and ANSI X9.52 (family X9 is standard for security in banking and financial applications).

Why not Double DES?

Double application of any encryption algorithms is subject to a **known-plaintext** attack named **meet-in-the-middle** which allows to decrypt data with at most 2^{N+1} attempts (if the keys are N-bit long). If the attacker knows one plaintext, it can create the attack. For this reason, the double version of encryption algorithms is never used, because the computation time doubles but the effective key length increases of just one bit.

Moreover, has been proved that, if the base symmetric algorithm is a **group**, then there exist an equivalent key K_3 so that:

$$enc\big(K_2, enc(K_1, P)\big) = enc(K_3, P)$$

It means that in this case the time needed for normal encryption/decryption doubles, but not even one single bit is gained in terms of K_{eq}.

Meet-in-the-middle attack

By hypothesis are used N bit keys and **known P and C** such that $C = enc(K_2, enc(K_1, P))$. Note that $\exists M$ such that $M = enc(K_1, P)$ and $C = enc(K_2, M)$. The attacker compute 2^N values $X_i = enc(K_i, P)$ and then compute 2^N values $Y_j = dec(K_j, C)$ and then there is the search of those values K_i and K_j such that $X_i = Y_j$. There can be "false positives", but they can be easily discarded if more than one (P, C) couple is available.

Let us assume a company protecting its communication between Torino and Milano by double DES. If the attacker sniffs the network (and reads the encrypted data) he can send a message over that line (which means that he is controlling the plaintext) and then he will see the ciphertext (of the plaintext sent) automatically generated from the security system. That is the way to get a pair of (P, C) without knowing the keys.

IDEA

While DES was free of charge, IDEA (**International Data Encryption Algorithm**) is patented with low royalty (only for commercial use, ASCOM AG). It uses **128-bits key** and **64-bits data block**. It is quite famous because it was initially used in PGP. It is suitable to be efficiently implemented via software because it uses three basic operations: **XOR**, **addition modulo 16** and **multiplication modulo $2^{16} + 1$.** By looking at these operations it comes up that the idea was considered for a specific class of CPUs, those with 16-bits size word, and this suggests that IDEA was developed for low powered CPUs, the ones employed in several places. An application of IDEA has been done many years ago in Formula 1 competitions, since it emerged that McLaren spied Ferrari by intercepting radio communication. Ferrari asked ASCOM AG to create encrypted devices, so they used IDEA encryption because it was suitable with radios due to the low CPU and battery consumption requirements.

RC2, RC4

These two kinds of cryptography were made by **Ron Rivest** and RC stands for **Ron's Code**. These were made specifically for its own company, so they are secret and proprietary of RSA (Ron's company). It is not patented because it is not disclosed. These algorithms from 3 to 10 times faster than DES. RC2 is a block algorithm while RC4 is a stream one, and both use variable length key.

RC2 was finally published as RFC-2268 in Mar 1998 and has been discovered that it uses from 8 to 1024 bits keys (usually 64-bits) and 64 bits data block. On the contrary, RC4 remains nowadays a secret algorithm. Since it is sold in form of library, someone people reverse engineering to make a possible C-version of the algorithm called **ARCFOUR.**

Application of block algorithms

"***How a block algorithm is applied to a data quantity different from the algorithm's block size?***"

There are two cases:

1. *Data size to encrypt > algorithm's block size*
   - **ECB** (Electronic Code Book): nowadays considered not secure, must never be used;
   - **CBC** (Cipher Block Chaining): this is currently the best way to apply an encryption algorithm to data that are bigger than the size of the algorithm's block size;
2. *Data size to encrypt < algorithm's block size*
   - **Padding:** to be used only when the data size is not exactly a multiple of one block size, for example when data is long 2.6 blocks. Note that this technique is not used to pad data that is just shorter than one block size;
     - **CTS** (CipherText Stealing): permits to use block algorithms without padding, so without increasing the size of the ciphertext with respect to the plaintext;
   - **CFB** (Cipher FeedBack), OFB (Output Feedback);
   - **CTR** (Counter mode).

ATTENTION! These are NOT algorithms; they are application modes for an algorithm

It is necessary to specify: the algorithm, the size of key and if it is a block algorithm also the mode of the application (e.g., AES-128-CBC).

ECB (Electronic Code Book)

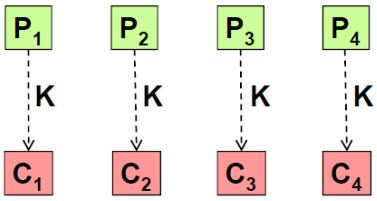


Figure 12 ECB mode applied to an algorithm: for each block $i = 0, \dots, N$, $C_i = enc(K, P_i)$. It is very fast since the computation of the cyphertext can be performed in parallel, but the same reason is source of insecurity

This mode splits the plaintext in blocks and each block is encrypted separately with the key; that means that, for each block $i = 0, \dots, N$, $C_i = enc(K, P_i)$. It must **NOT** be used! [*Lioy said that the exam will not be passed if ECB is used somehow*]. In particular, it must not be used on long messages (any message longer than 1 block) because:

- If the attacker is intercepting the cyphertext and exchange the position of 2 blocks, it is not detected (**block swapping)**. This permits to exchange data inside an encrypted message;
- **Identical blocks generate identical ciphertexts**, hence it is vulnerable to **known-plaintext** attacks. Plaintext attacks require the precomputation of all possible encryptions of a known plaintext: comparing the encrypted block with all the precomputed possible encryptions, it is possible to figure out what the key is.

Known-plaintext attack example

Let us suppose that we want to intercept a message sent by the rector of the Politecnico di Torino. It is possible to assume that his messages will contain the word "Torino", so we start encrypting this word with all possible keys. Next, we can go sniffing the blocks sent by the rector: if any of these blocks is equal to the encryption of "Torino", then we can use the key used to obtain that encryption of "Torino" to decrypt the rest of the message. Some arguments against this method could be: in which way is "Torino" written (e.g., "TORINO", "Torino")? What if "Torino", since it is shorter than one machine word, is split in different blocks (e.g., P_i= "…To", P_{i+1}= "rino…")? The known-plaintext attack is made possible by the fact that usually people exchange data in a *structured format*, so that there are metadata that are usually known (e.g., fixed header).

If a Word file is created with just a word, it is big KBs of memory (and not just bytes). This happens because Word applies a header on the document which is always the same (and always at the beginning of a file). If the first block of a Word file is encrypted, it can serve as dictionary for Word files. This means that we do not even have to guess some plaintext, but just the file format.

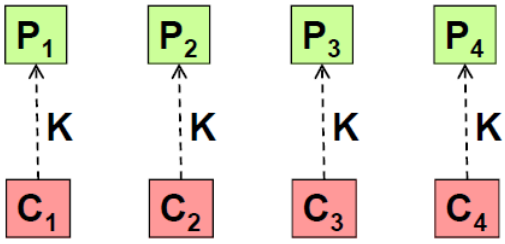


Figure 13 ECB mode applied to an algorithm: decryption is simple, for the i_{th} block it is $P_i = enc^{-1}(K, C_i)$.

Decrypt

The decryption is simple because it is needed to get the cyphertext block and decrypt it with the key. An error in transmission generates an error at the decryption of one block (**no propagation** of the error on other blocks).

CBC (Cipher Block Chaining)

This mode resolves all ECB's problems. It splits the plaintext in blocks but each block, before being encrypted, is XORed with the cyphertext of the previous block. This makes encryption unpredictable. On the first block (which is the header of the file mentioned before) there is a problem. To apply CBC, it must be added an element called **IV (Initialization Vector)** that be considered as C_0 that has

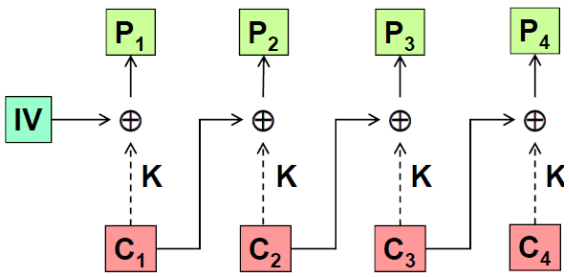


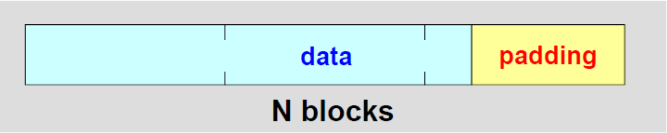
Figure 14 CBC mode applied to an algorithm: for each block $i = 0, \dots, N$, $C_i = enc(K, P_i \oplus C_{i-1})$, with $C_0 = IV$.

the only purpose to alter the first plaintext block. There is also protection against block swapping because if some blocks are swapped the output will be different, because they will be altered with the wrong element of XOR.

Decryption

The IV vector needs to be known also at the receiver because when the cyphertext is received, to perform decryption with the key the IV must be known to decrypt the first block (since XOR is the inverse operator of itself). IV must also be different every time (hopefully random, to avoid being guessed) because its purpose is to make impossible to precompute the possible encryptions of the first block. It must also be a **NONCE** (**Number used once**) which means that IV should be generated and never re-used in the future. IV in some cases is sent in clear because even if the attacker knows it, he could start the computation only at that moment, and this is a long task; moreover, it will permit him to attack only the sent message, because the next one will have another IV. IV can also be sent encrypted by using ECB since that it is just a block.

Padding (aligning, filling)



It is not always possible to have the plaintext split in blocks that have all the same dimension. In the picture data is made of two blocks and a piece that is not a full block (small part at the end). To encrypt the small part the technique is called **padding**. Some bits are added at the end of the original data until the multiple of block is reached.

It has some problems since:

- **The cyphertext gets longer than the plaintext**: adding useless data means transmit/store more data than needed;
- Which should be the **value of padding** bits? How can we distinguish the padding from the original plaintext at the receiver?

Padding Techniques

- If the length is known or it can be obtained as in the case of a C string it is possible to add **null bytes** at the end **0x00 0x00 0x00;**
- The original DES specified to add one "1" bit followed by as many "0" as needed **1000000:**
- One byte with value 128 followed by null bytes **0x80 0x00 0x00**
- Last byte's value equal to the length of padding **0x**??**0x**??**0x03** (3 bytes of padding. The last one has value 3 and previous has value to be specified)

There are so many kinds of techniques because the padding technique selected will permit to make or avoid different attacks.

There are many kinds of values that can be selected (d*o not remember all the techniques, just remember there are many and the proper one must be selected)*:

- **(Schneier) null bytes** $e.g.\ 0x00\ 0x00\ 0x03$
- **(SSL/TLS) bytes with value L** e.g. $0x03\ 0x03\ 0x03$
- **(SSH2) random bytes** e.g. $0x05\ 0xF2\ 0x03$
- **(IPsec/ESP) progressive number** e.g. $0x01\ 0x02\ 0x03$
- **Byte with value L-1** e.g. $0x02\ 0x02\ 0x02$

Some notes

- B stands for the size of algorithm's block
- D stands for the size of data to process

Some of these techniques offer (minimal) integrity control: if key is wrong or data is manipulated, then the padding bytes are incoherent (e.g., length of something bigger than one block or wrong padding values).

Typically, this is applied to large data, on the last fragment resulting from the division in blocks (e.g., for ECB or CBC). If $|D| < |B|$ an ad-hoc technique is preferred (CFB, OFB, CTR, …). If there is just a byte it is not good to add 65 bytes of padding.

Even if the plaintext is an exact multiple of the block, padding must be added anyhow to avoid errors in the interpretation of the last block. The biggest padding is required when there is no padding to add.

With SSH2 padding equal data give different ciphertexts (even when encrypted with the same key).

The padding type for a certain algorithm determines the type of (some) possible attacks, but it also depends upon the algorithm used.

Ciphertext stealing (CTS)

When using padding, the size of the cyphertext is bigger than the plaintext. This may not be acceptable to several cases (for example data encrypted on hard disk that may not fit in the same disk). It is possible to use CTS instead of padding. The last (partial) block is filled with bytes taken from the second-to-last block (encrypted), then these bytes are removed from the second-to-last block (which becomes a partial one). After the encryption, the position of the last and second-to-last blocks is exchanged.

It is useful when it is not possible to increase the size of data after encryption, but the computation time slightly increases.

CTS Example with ECB (encryption)

1) The plaintext is split in blocks;
2) The second-to-last block is full, so it is possible to encrypt it with the key;
3) The last one is not full and create a problem;
4) The encrypted block C_{n-1} is split in two parts: **head** and **tail**. Tail has the same size as the part that is missing in the last block;
5) The **tail** is added in the partial block, which creates a full block;
6) The full created block is transmitted, and then the *head* is transmitted.

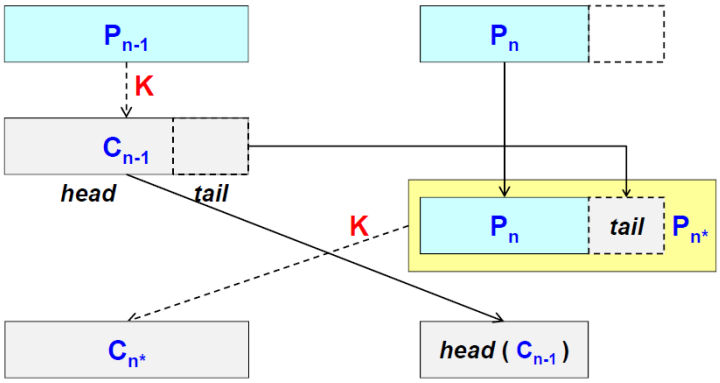


*Figure 15 CTS Example with ECB (encryption): please note however that **ECB MUST NEVER BE USED**; the example has the purpose to make easier to understand CTS*

The receiver must invert the procedure. Bits corresponding to the tail are encrypted twice.

Exchanging the blocks is needed because if we send the head without the tail, then the beginning of the next block is interpreted as part of the previous one.

CTS Example with CBC (encryption)

The various blocks are encrypted with key and XOR, in particular C_{N-2} which is used in the P_{N-1} XOR to encrypt. In the last step P_N is padded with all zeros and it is XORed with the encryption of the block $N-1$ and in this way the last block is encrypted. E_N is placed in the $N-1$ position. On the contrary, the E_{N-1} encryption is not transmitted completely, but only the **head**. It seems that part of $E_{N-1}$is missing, but the since the tail is being XORed with all zeros will be recovered by C_{N-1}.

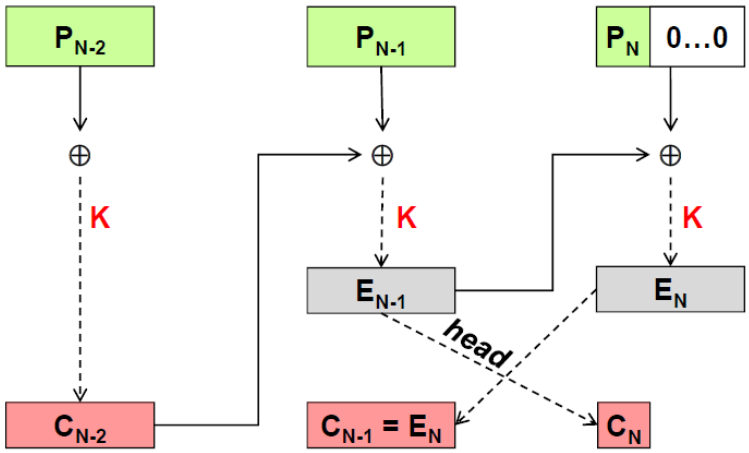


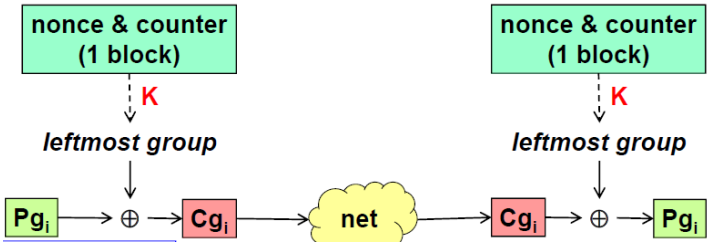
Figure 16 CTS Example with CBC (encryption).

Since XOR with zero does not alter data, then the tail is embedded into E_N as the $N-1$ block. In this way the cyphertext has the same dimension of the plaintext. With this mode, if the blocks are fully filled from the plaintext, there is no padding needed to add. It is nowadays very much used, especially for encryption of storage on embedded devices (e.g., smartphones).

CTR (Counter mode)

This is useful for plaintext smaller than one block and it does not make sense to encrypt it using padding. CTR is the most used technique that uses a block algorithm to cipher N bits at a time (a "group", often a byte). It has nice properties that permits random direct access to any ciphertext group.

Considering CBC, it is not possible to encrypt or decrypt any block (because concatenation is needed), but here it is possible to decrypt one single block, but it requires a nonce (also called IV) and a counter, that are in some way combined (concatenated, summed, XORed, etc.).



In the example there is a group (for example 1 byte) smaller than 1 block called Pg_i and on the top there is a register which is exactly big as one block. The register is filled with nonce and counter, composed in the way we decided. Since it is one block, we can directly encrypt it with the key. Then the leftmost group is taken. For example, if Pg_i is 1 byte, only the last byte will be taken from the encryption of the register. Then, we will apply XOR that will generate the corresponding cyphertext Cg_i that can be sent through the net.

On the right, the receiver must have another register initialized in the same way as the sender and do same inverse operation. Of course, sender and receiver must have the same nonce and the same value in the counter (they must be sync). This kind of transmission is sensitive to **cancellation attacks**, because if a message is removed, there will be no more synchronization. It can be avoided with integrity techniques (discussed later).

To sum up, advantages of this technique are:

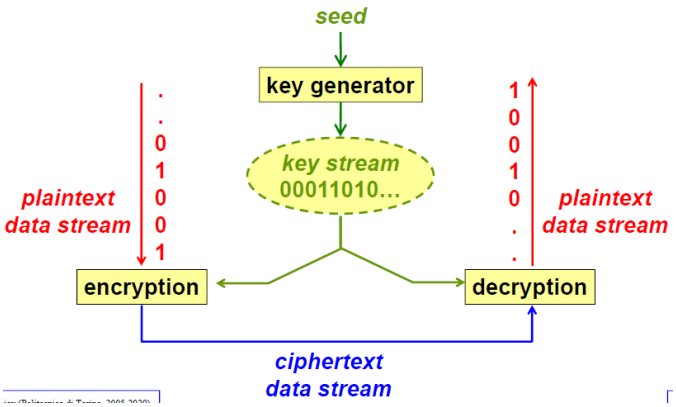
- **Direct access to any group**: it is possible to decrypt one group at will, without first decrypting any other;
- **no propagation** of the error on other groups: if there is an error during transmission, it will affect only the corresponding plaintext group;

Usually applied when it is needed to encrypt a few data, counter mode can be used to encrypt large data by encrypting some bytes per time.

Algorithms of type stream

Stream algorithms do not require the division of the plaintext in blocks. They typically work with 1 bit or byte at a time. In this category there is just one perfect algorithm which is called **one-time pad** and it requires a key as long as the message to protect: for this reason, it has no practical use.

Real algorithms use pseudo-random key generators, which must be synchronized between the sender and the receiver (for example RC4 and SEAL).



The plaintext data stream on the left is encrypted one bit per time. Since the data stream can be long, the key stream must be long too. The key stream is generated from a key generator, which is initialized with a **seed** (the seed is actually the key, since it must be the same for receiver and sender). The encryption function will basically be a XOR operator (no more complex function is required) because the operations will be unpredictable. At the receiver, the decryption will be applied by using the same key stream which means to use the same seed and the same algorithm for generating the key.

If the attacker can delete part of the cyphertext data stream the decryption will come out wrong since the decryption will use a wrong part of the key stream. On the other hand, these algorithms are really fast.

Curiosities

The importance of stream algorithms is highlighted from the international competition named **ESTREAM** that asked cryptographers to submit their best stream ciphers for evaluation. It has been completed on April 2008 but revised periodically (last revision on January 2012). They selected an algorithm portfolio (depending if you want to implement on hardware or software):

- (software, 128-bit key) → *HC-128, Rabbit, Salso20/12, SOSEMANUK*
- (hardware, 80-bit key) → *Grain v1, MICKEY 2.0, Trivium*

Extended versions:

- (software, 256-bit) *HC-256, Salsa20/12*
- (hardware, 128-bit) *MICKEY-128 2.0*

Salsa20 and ChaCha20

These are symmetric stream algorithms invented by D. J. Bernstein which have 128 or 256-bit keys. ChaCha20 is an improvement of Salsa20 where there's more "diffusion" of bits (which means that if you change 1 bit of the input, more bits of the output are changed) and is faster on some architectures.

The base operation is a 32-bit ARX (add – rotate – XOR), while the base function is:

$$f(\boldsymbol{Key256}, \boldsymbol{Nonce64}, \boldsymbol{Counter64}) = \boldsymbol{512\ bit\ keystream\ block}$$

This permit $O(1)$ decryption of any block at random. Salsa20 performs 20 mixing rounds of the input, while Salsa20/12 and Salsa20/8 have a reduced number of mixing rounds: 12 and 8 respectively, which is faster but less secure.

ChaCha20 has been standardized and is heavily adopted, with documentation RFC-8439 (Chacha20 and Poly1305 for IETF protocols). The IETF has slightly modified the original specification: bigger nonce and smaller block counter. In particular,

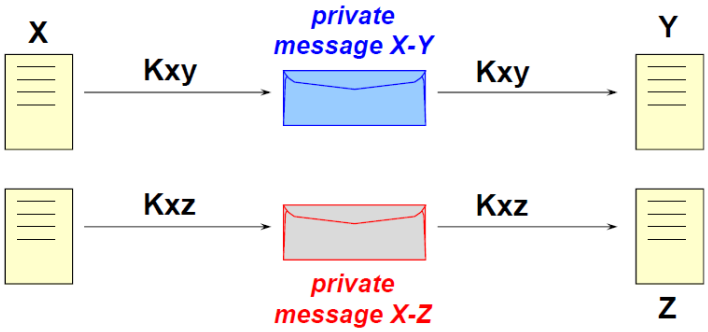
- 96-bit nonce;

- 32-bit block counter;
- Hence a limit of 256 GB (2^{32} 64-byte blocks);
- To overcome this limit, use the original definition by Bernstein.

This has become famous because Google is using it (for Chrome on Android), openssh and various CSPRNG (e.g. /dev/urandom since Linux kernel 4.8).

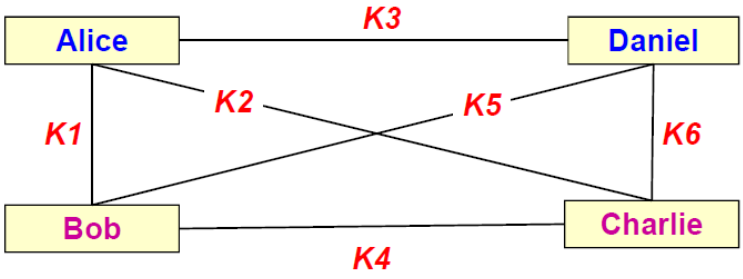
Symmetric encryption

This type of encryption uses a single and secret key. It is needed one key for each couple (or group) of users. In the example X has a shared key with Y. If X wants to talk with Z, they will need a different key. But it is also possible to decide to have a common key (group key) which is common to X,Y and Z but other people that are part of the group, but are not the destination of a message, would be able to read it.



Key distribution for symmetric cryptography

A complete pairwise private communication between N parties requires $\frac{N\times(N-1)}{2}$ keys, which is equal to the diagonals of a polygon with N vertices. If N gets bigger, the number of keys gets much bigger too. The problem is how to give securely the keys to the communicating parties. The solutions are:



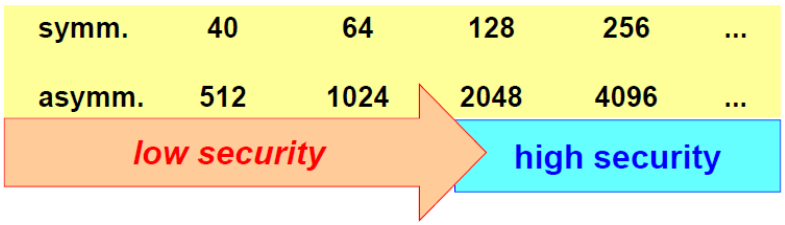
- **OOB** (Out-Of-Band): the key is given to the parties not using the same electronic channel used for transmitting the message (direct communication would be the best solution for key sharing, e.g., whispering the key to someone);
- **By means of key exchange algorithms**: we will discuss this later.

Length of secret keys

The length of secret key should be adequate. Adequate means that if:

- the encryption algorithm was well designed and
- the keys N_{bit} in length are kept secret;

then the only possible attack is the **brute force (exhaustive) attack** which requires $T = 2^{Nbit}$ trials (number of possible combinations). The table on the right summarize what is currently considered secure or insecure. Any key less than 128-bit is **unsecure**. This is due to current CPU's speed and the fact that it is possible to parallelize the research of the key (e.g., several computers that work on a specific set of keys). The second row refers to **asymmetric key.** Every asymmetric key under 2048-bits is unsecure. There are no borders in the column but an arrow that is always expanding to the right. The area of low security increases by year because every year computers are more powerful. Another problem is **for how much time will data remain secret.** If secrets have to remain as such, it is important to



avoid the low security arrow and always stay in the right part of the table, but at the same time the more the key's size increase, the slower the algorithms become and it's not simply doubling. For symmetric algorithms doubling the key cuts in half performance, but for asymmetric algorithms doubling the key means a factor from 4 to 10 of lowing performance.

DES challenges

The RC2 and RC4 were private sold algorithms, but the creator did not achieve a lot of money. For this reason, the creator started a campaign to demonstrate that DES was insecure. He put a price of $10.000 for the 1st person that was able to decrypt a DES encrypted message by using brute force. It means $2^{56} = 72.057.594$ billions of possible keys.

DES challenge I started on 18-feb-1997 and ended on 17-june-1997. After 4 months someone discovered the encrypted message by using 15.000 computers in network and after trying about 25% of keys, 17.731.000, the key was found.

DES challenge II started on 13-jan-1998 and ended on 23-feb-1998. Someone discovered it in a month by using 20.000 computer by using 87% (63.686.000) possible keys.

It is not the number of computers used that made the difference, but the power of computer that in 1 year has changed.

Des challenge III started on 13-jul-98 and ended on 15-jul-98. In two days, 25% (17.903.000) of keys was tried. It means that in just a week it was possible to decrypt any message with DES encryption. This time only 1 special-purpose system (DEEP CRACK) was used, developed by the EFF at a cost of $250.000. They created this special purpose to demonstrate to the government (that was using DES) that it was unsecure.

This is the demonstration that it is possible to construct a computer system that can decrypt a generic DES message, but:

- It is necessary to know the type of data (e.g., ASCII), otherwise it is needed to try all different kind of formats for files. (in the competition it was known that it was a text message written using ASCII);
- The machine cannot decrypt 3DES messages;
- DES is not intrinsically weak; it only uses a short key.

DES challenge IV started on 18-jan-1999 and ended after 22 hours. The 22% of keys was tried by using the DEEP CRACK machine plus a few thousands of workstations. The peak power was 250 Gkey/s while the average power was 199 Gkey/s.

After these challenges IETF changes all RFC advising not to use DES and suggesting the use of 3DES. There is a specifically RFC-4772 that contains the security implications of using DES. A German bank sentenced for a fraud made by means of a system exploiting the weakness of DES. On 15-jan-1999 FIPS withdrawed DES (46/2) and replaced it with 3DES (46/3). At the same time US government started a competition to select a new symmetrical algorithm: **AES (Advanced Encryption Standard)** with key length up to 256 bits and block size at least 128 bits. There were 15 candidates but only 5 finalists.

The winner was announced on 2nd Oct 2000, called RIJNDAEL and the algorithm was published in November 2001 as FIPS-197. Two notes for this algorithm:

- Not the best for everything, but the best for many kinds of systems (e.g., embedded systems).
- RIJNDAEL was the only algorithm in which America was not involved (to avoid any accusation of manipulation).

Even if the algorithm was published in 2001, it has been gradually adopted after 2010. It takes so long because crypto algorithms are like wine: the best ones are those aged for several years.

Public key / asymmetric cryptography

Nowadays there is a second kind of cryptography invented recently (‘70s of last century) and it is named **public cryptography**: in this kind of cryptography, **key-1 ≠ key-2**. If one is used for encryption, the other one must be used for decryption, but the role can be exchanged. It is also called **asymmetric algorithm** and the keys are used in **pairs**. Since there is not only one key, the keys are named according to the way in which they are stored: one is kept private (**private key**), while the other one is public (**public key**). This kind of cryptography has a high computational load, so it is usually used to distribute secret keys and to create electronic signatures (with hashing) and not for storing data. Main algorithms are: *Diffie-Hellman, RSA, DSA, El Gamal, …*

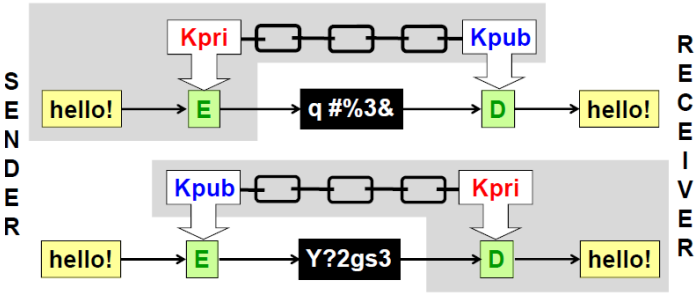


*Figure 17 How a key pair can be used in asymmetric encryption in a communication: keys are generated in **pairs**: **private key** (Kpri) and **public key** (Kpub). Keys have **inverse functionality**: data encrypted with one key can be decrypted only by the other key.*

On the top, a message is encrypted with a private key and sent to the receiver, that will use the corresponding public key to decrypt the message.

Typically, it is the sender who use its private key to encrypt something. All the other people in the world will know the public key and will be able to decrypt the message.

On the bottom, there is the case in when someone is using the public key of the destination, because only the destination will have the corresponding private key.

These two ways of using a pair of keys is used to obtain two different functionalities, described more in detail below.

Digital signature

The following is just the **idea** for digital signature. Digital signature follows the top case in the previous image, the encryption of data made with the private key of the author. X encrypts a document with the private key of himself. Since it was encrypted with the private key, anybody can use the public key to decrypt it. So, this is not a secret message, but it is “signed” by the sender: if someone use another public key it will not be possible to decrypt the message. This is a proof and is legally valid that some bits were created by a person. If the message is too big it will take too long to encrypt the message: for this reason, data is not directly encrypted but only its summary (*digest*) is. This provides **data authentication (and integrity)**.

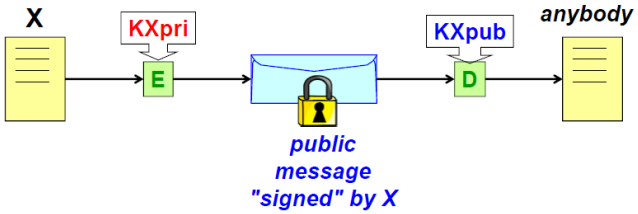


Figure 18 Using asymmetric encryption for signing a message

However, this is NOT the real implementation of the digital signature, that is completely explained next.

Confidentiality without shared secrets

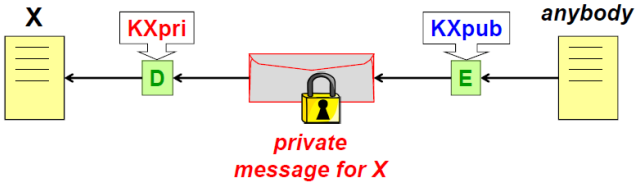


Figure 19 Using asymmetric encryption to secret a message

It is possible to generate a **secret message** for a receiver given only its public key. The message is encrypted with public key and only the receiver will be able to decrypt it using the private key. It does not need to share the key between sender and receiver. If it is a big message it would take so much time to encrypt/decrypt.

Public key algorithms

There are two mains algorithm used nowadays: **RSA** (*Rivest – Shamir – Adleman*) and **DSA** (*Digital Signature Algorithm*).

RSA consists in the computing of the **product of two prime numbers**. If you are the attacker, you see the result and if you want to attack the system you must discover what are the two factors. Therefore, very long keys are needed, because if the result is 21 it will be easy to attack: 3 and 7 will be the prime numbers. The point is that there are no formulas to compute all prime numbers, for this reason the bigger is the number you are considering, the harder finding the prime numbers is, so the more complex is the problem for the attacker. RSA can be used to implement both functionalities: secrecy and digital signature. It is patented only in USA by RSA, but the patent expired on 20/09/2000 (no problems of royalties).

DSA is the main competitor for digital signature only. This algorithm exploits a similar mathematical problem: **given a base and an exponent, perform the power**. If you are the attacker, you see the result, but it is not easily known which is the base and the exponent of that number. This is like taking the logarithm of a number without having the base. The problem of DSA is that it can be used only to create a digital signature, because inside the algorithm there is a one-way lossy compression function: losing information, the original plaintext cannot be recovered. That was done on purpose because DSA was designed by US government and they did not want to give citizen the ability to hide something. To have confidentiality without shared secrets you must use the **El-Gamal algorithm** (with the same principle of DSA). DSA is a standard NIST for DSS (FIPS-186).

RSA computational optimization

Usually, all public keys have $E = 3, 17\ or\ 65537$ ($0x10001$, the Fermat number). "E" stands for exponent and those number are selected because they have only two bits with "1" and all the others at "0", because the more bits are raised to 1, the more operation must be done. In this way we get high speed of encryption operation (when making confidentiality without sharing secrets) and high speed in signature verification. The downside is that the other operations are slot (decrypting, creating signature). Since these values are common, some companies started developing algorithms optimized for these special cases, but the consequence is that the algorithm is deoptimized for all other values of the exponent.

This led two famous attacks against Microsoft because they used one of these "special" algorithms and the attack created a signature that had a lot of 1s in the exponent and then gave it to Microsoft Server (and this made a sort of loop at the server, because computing takes so long).

The appropriate length of public keys is 2048 bits that offer an appropriate security level for several years. This is proved by means of RSA challenges with some prizes. In 2012 MS applied a patch to their systems to do not accept any more RSA keys < 1024. The same for Mozilla, that from 2013 does not accept RSA < 2048 & MD5.

RSA challenges

- **Solved challenges (old style, size in decimal digits)**
  - 10-apr-1996, RSA-130, 1000 MIPS-years
  - 22-aug-1999, RSA-155 (512 bits), 8000 MIPS-years
  - 9-may-2005, RSA-200 (663 bits), ~75 years Opteron 2.2 GHz
  - 28-feb-2020, RSA-250 (829 bits), 2700 core-years Xeon 2.1 GHz
- **Solved challenges (new style, size in bits)**
  - 3-dec-2003, RSA-576 (174 decimal digits)
  - 2-nov-2005, RSA-640 (193 decimal digits)
  - 12-dec-2009, RSA-768 (232 decimal digits) ~ 1500 years Opteron 2.2 GHz; record on 5/10/2019
- Prizes withdrawn after 2007, interest moved towards other cryptographic problems

Twinkle

Every year there is a conference named Eurocrypt, which is a conference about cryptography in Europe. There was a scientist from RSA Laboratories that attended the so-called Ramp Session (means that you show some interesting and promising ideas). Professor Adi Shamir (who gave the "S" in RSA) showed an unusual piece of hardware called "**TWINKLE**" (which stands for **T**he **W**eizmann **I**nstitute **K**ey **L**ocating **E**ngine), which is an electro-optical computer that implement the sieving algorithm to find prime numbers. This Twinkle machine

should be able to find the factors of an RSA key between two or three orders of magnitude as fast as a conventional fast PC. It runs at a very high clock rate (10 GHz), must trigger LEDs at precise intervals of time, and uses wafer-scale technology. The estimates of the professor are that the device can be fabricated for about $5.000. Shamir never got back to the next conference to show the device.

Twirl (The Weizmann Institute Relation Locator)

This is an electronic device for factoring of large integers. It implements the sieving step of the Number Field Sieve integer factorization algorithm, which is in practice the most expensive step in factorization. TWIRL is more efficient than previous designs by several orders of magnitude, due to high algorithmic parallelization combined with adaptation to technological hardware constraints. Although fairly detailed, the design remains hypothetical since the device has not been actually built. TWIRL is built using current VLSI technology and it will be possible to factor 1024-bit integers, and hence to break 1024-bit RSA keys in 1 year at the cost of a few dozen million US dollars.

Firmware signature for TI calculators

These facts demonstrate that RSA is not totally secure: there are evidence that sooner or later the factors will be found the mathematical problem solved. An example is the TI83+ graphing calculator, which is a graphing calculator that has a firmware protected by 512-bit RSA signature. Signature key has been factored on July 2009 after 73 days of computation on a 1.9GHz dual-core Athlon64 PC. On 1999 the same computation would have required 8000 MIPS-years + Cray C916. Now all users may autonomously modify the firmware by themselves. A signed firmware mans that a software is signed with a digital signature (which is a piece of software). If anybody alters the software, then the digital signature will fail and there is a hardware control that will not operate if software is modified.

Key distribution for asymmetric cryptography

Private key must be never disclosed and must be protected. The other key does not need protection, since it is a public key must be distributed as widely as possible. But the problem is: ***who guarantees the binding (correspondence) between the public key and the identity of the person?***

There are two solutions for this problem:

1) **Exchange of keys OOB** (Out-Of-Band) (e.g., key party!)
2) Distribution of the public key by means of a specific data structure named **public-key certificate (= digital certificate)**. But which is the format of certificate? Do you trust in the certificate issuer?

Secret key exchange by asymmetric algorithms

In principle asymmetric cryptography can guarantee confidentiality without shared secret by itself but, as previously said, it is slow, and this makes it applicable to small quantity of data. For this reason, a common practice is to use it to send the secret key chosen for a symmetric algorithm, solving at the same time the issue of symmetric key distribution of symmetric algorithms and slowness of asymmetric ones for big data.

In the example figure, X is willing to send an encrypted message to Y: to do so, it produces a key K (long usually 128-256 bits) and encrypts it using the public key of Y (KYpub), resulting in ε: note that, given the small size of K, performing its asymmetric encryption is fast enough. At this point Y uses its private key to decrypt ε, obtaining the original symmetric key K, that will be used for the rest of communication between X and Y, in the example to send "hello". Typically, the key K can be changed for each message or for each network session; however, it is not meant to be long term (e.g., years). This type of encryption is applied every time a server connection is created.

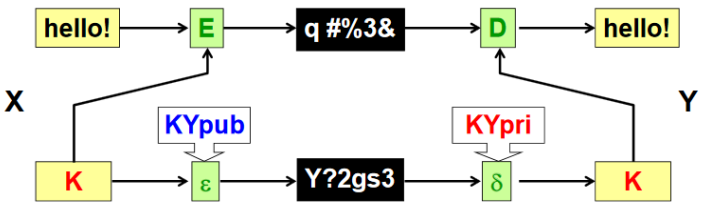


Figure 20 Using asymmetric encryption to share a symmetric key with confidentiality

Is this schema issue-safe? In some environment the fact that X decides what key will be used for the actual communication is not acceptable: Y could argue that X's key is not safe, maybe it has been disclosed and someone could be able to decrypt the message in the communication even not violating the asymmetric encryption used to exchange the key. The same reasoning could apply to the opposite case, in which Y decides the key to be used. In such cases, the key can be produced jointly using the **Diffie-Hellman** algorithm.

Diffie-Hellman algorithm

The two parties (called X and Y in this example) involved in the communication agree about two public integers p (a large prime number) and g (called "generator"), given the constraint $p > g > 1$. The length of a Diffie-Hellman key is defined as the number of bits in the binary representation of p. Both numbers p and g are public values and typically $g = 2, 3\ or\ 5$. Both parties decide autonomously a (random) number, we call $\boldsymbol{x}$ for peer X and $\boldsymbol{y}$ for peer Y: X computes the number $A = g^x mod\ p$, and Y computes $B = g^y mod\ p$, then they exchange these values in clear. When the values from the other party is received, X computes $K_A = B^x mod\ p$ and Y computes $K_B = A^y mod\ p$: it is easy to see that $\boldsymbol{K_A = K_B = g^{xy} mod\ p = K_{AB}}$, so the parties have decided together a value of the symmetric key to be used, because it depends from both *x* and *y*.

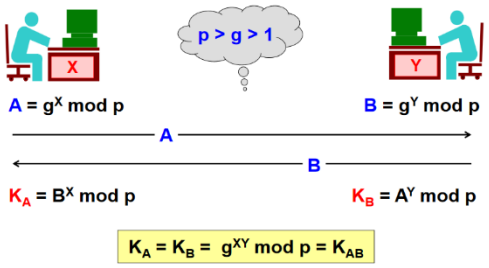


Figure 21 Agreeing on a symmetric key using Diffie-Hellman algorithm

From an attacker's perspective, even knowing the values for p and g, so the values A and B being exchanged (e.g., trying to perform a **passive** *man-in-the-middle*), it is not possible to compute K_{AB}, since he would need to also know *x* or *y*. So Diffie-Hellman algorithm is resistant sniffing attacks; however, it is vulnerable to *man-in-the-middle* attacks if the attacker can manipulate the data: this manipulation can be detected if the values being exchanged are authenticated. In this case, certificates for Diffie-Hellman keys are needed, or there is a variant of the algorithm called **Authenticated Diffie-Hellman** (or **MQV**, from the names of the scientists who invented it, Menezes-Qu-Vanstone).

Since it uses only public information, it is referred as the first public key algorithm invented, and it is frequently used to agree on a secret key and so it is referred as a **key-agreement algorithm**, as opposed to **key-distribution algorithm** just seen. It was patented in the USA, but it is free of royalties from 29 April 1997.

DH: man-in-the-middle attack

As said, the only way to attack the Diffie-Hellman having performed the sniffing, is to run a brute-force attack to find $x = \log_g A\ mod\ p$, that is infeasible to solve since it is a discrete logarithmic problem. In the case of an attacker performing active man-in-the-middle the outcome is different.

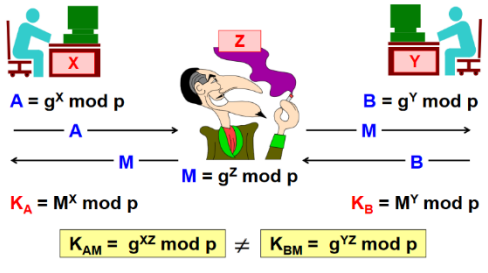


Figure 22 An example of an successful active man-in-the-middle attack to a communication using Diffie-Hellman algorithm

Let us suppose that a man in the middle Z is participating in the communication so, like X and Y, it computes $M = g^z mod\ p$: when peer X sends its value A, Z stops it and send the value M in exchange. Analogously, when peer Y sends its value B, it stops it and sends the same value M. By doing so, peer X will compute $K_A = M^x mod\ p$, while Y will compute $K_B = M^y mod\ p$: this time the two values are not equal, in fact $K_{AM} = g^{xz} mod\ p$ while $K_{BM} = g^{yz} mod\ p$. They are actually computing a shared key with the man in the middle, and they will encrypt data that can be decrypted by the man in the middle, who will then inspect the data, modify it and then will encrypt it again for the destination, with the other shared key.

This attack is made possible because when a peer receives a value, it has not any proof of who sent it: this is the reason because one possible solution is to use certificates. One application of DH could be in HTTPS: the browser could produce a key and send it encrypted with the public key of the server, or the server could publish

its DH value signed with its private key and distribute it. DH is much used nowadays to agree on a key and there is also the possibility to also have authentication (because otherwise such MITM attack is possible).

Brute-forcing Diffie-Hellman: cryptography against quantum computers

DH has the complexity of the discrete logarithm, the best way to find the solution $x = \log_g A \ mod\ p$, is to perform successive products, so the complexity is linear in p, hence exponential in the number of bits of p. There are also other algorithms, but still they are not polynomial in time.

However, it is possible to use the Shor's algorithm, whose complexity is $O(\log(N)^3)$, that is capable to solve RSA and DH, but requires a quantum computer. Quantum computing is at the moment an experimental technology: in 2012 a quantum computer with 7 qubits computed the factorization of $21 = 7 \times 3$, and in 2014 an adiabatic quantum computer factored $56153 = 233 \times 241$. These results are remarkable, but still far enough from the large numbers used nowadays. Since signatures and secrets must resist for many years (about 10-30 years), there are reasons to be worried about quantum computers becoming powerful enough to break these security mechanisms: experts are starting to look for other mathematical problems that cannot be attacked by a quantum computer.

A new kind of cryptography of this type is **ECC** (*Elliptic Curve Cryptosystem*). The concept is the following: instead of using modular arithmetic, the operations are executed on the surface of a 2D (elliptic) curve. Not all the points in the cartesian space are valid points, but only those who satisfies the 2D curve equation. The problem of the discrete logarithm on such a curve is that since the point representation is more complex, then in general the problem is more complex than the one in normal modular arithmetic. This gives the opportunity to use keys shorter to 1/10. Most of the algorithms have been revisited to adapt them to the elliptic curve (RSA cannot be adapted):

- **ECDSA** for digital signature:
  - message digest computed with a normal hash function (e.g. SHA-256);
  - signature = pair of scalars derived from the digest plus some operations on the curve (see below about operations on the curve).
- **ECDH** for key agreement;
- **ECMQV** for authenticated key agreement;
- **ECIES** (EC Integrated Encryption Scheme) for key distribution:
  - generates a symmetric encryption key (e.g. an AES-128 one) with operations on the curve;
  - gives to the receiver the information (based on his public key) needed to recompute the encryption key.

How does it work

The first element to be selected in a EC algorithm is the curve itself, that must be of the form of $y^2 = x^3 + ax + b$ with $4a^3 + 27b^2 \neq 0$, then two points P, Q belonging to the curve are selected, then the point $R = (x, y) = P + Q$, with $x = \lambda^2 - x_p - x_Q$ and $y = \lambda(xP - x) - yP$, where:

- $\lambda = (y_p - y_Q)/(x_p - x_Q)$ $if\ P \neq Q$ so it can be used to compute P+Q;
- $\lambda = (3x_p + a)/2y_\rho$, $if\ P = Q$ so it can be used to compute 2P.

Given these formulas, only addiction of two points and multiplication of a point by a scalar are defined and used by EC algorithms.

EC-Diffie-Hellman

With reference to the original DH, now the peers do not agree on p and g, but they agree upon the same elliptic curve (same set of parameters a and b), and a point G belonging to the curve. Then A chooses a random value x and computes $X = xG$; similarly, B computes $Y = yG$. Then A and B exchange these values, and each performs the multiplication of the value received by the value randomly chosen before, so A computes $K = xY$ and B computes $K' = yX$. As in the simple DH, it holds that $\boldsymbol{K = K' = xyG}$. So, the operation is simpler

than in simple DH, but the complexity for the attacker relies on the computation of values x or y knowing X or Y.

Case History: Sony PS3 hacking (Console Hacking 2010, PS3 Epic Fail)

The Sony PS3 is a Linux machine on which the direct access to system is blocked by Sony as well as any modification to it by digitally signing the firmware by Sony itself. PS3 has embedded Linux with loader verifying the binaries' ECDSA signature before execution. Generation of an ECDSA signature requires a random nonce, otherwise from the signature the private key can be computed. The problem was that Sony has decided a *fixed* random number (so technically is not a nonce anymore) so, as a consequence the private key was computed and distributed world-wide, so that anybody can run his own binaries on his PS3.

Sony tried to take the hackers to court, accusing them of violating their software: the judge dismissed the charge because the security measure was not implemented correctly.

Message integrity

So far, we have seen that there are methods to guarantee that a message from a sender to the receiver cannot be understood by a third party. However, it is still possible for an attacker to intercept the traffic from the two parties and change the encrypted message: even if this does not give to him the possibility to alter the message in a specific way, he can change it in an unpredictable way (for example flipping some bits in the message). In certain type of communication this causes the message to become not intelligible anymore when decrypted by the receiver, so that it is evident that something went wrong during transmission: in this case the receiver can ask for retransmission, and a continuous attack like this is at least a DoS. In other cases, for example when the message is a bank account number, there is a high likelihood that the result will be a different number, but still a valid one, so the destination has no way to check if it is wrong. This is particularly dangerous when the communication takes place between automatic systems.

So, along with confidentiality, integrity is a very important matter. Integrity is not about preventing modifications of data, it means that *when some data is received, the receiver can verify if the data has been modified or not*. This is important not only in communications, but also for storing data on the disk: can be important to verify is someone has changed the data on the disk since it has been written. The importance of integrity is shown by a study conducted by UK army, that estimated that only 10% of their communications needed confidentiality, but 100% of them needed integrity. **Integrity also implies authentication** since a message that has not been modified is also authentic (which means that is the same sent by the sender).

To guarantee integrity the used technique is to compute a ***digest***, that is a *fixed-length summary* of the whole message to be protected. A digest is conceptually similar to a checksum (widely used in communication to detect transmission errors); however, the algorithms used to compute a checksum are easy to attack, so a digest is usually calculated via a **cryptographic hash function**. The digest must be:

- Fast to compute;
- Impossible or very difficult to invert, that is go back from the digest to the original data;
- It should be difficult to create “collisions” (two different messages should not produce the same digest).

Cryptographic hash functions

The hash function works like this:

- split the message M in N blocks $M_1 \ldots M_N$;
- iteratively apply a base function (f);
- $V_k = f(V_{k-1}, M_k)$ with $V_0 = IV$ and $h = V_N$.

The IV is an initialization value which does not need to be announced or be random. Typically, it is fixed and specified inside the algorithm itself (source and destination must perform the same computation)

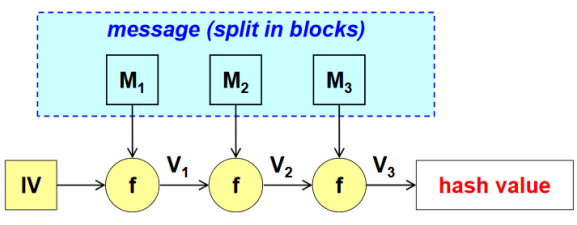


Figure 23 How a cryptographic hash function works

SHA-2 family

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

The most widely used algorithms nowadays are those ones in **SHA-2 family**: they have the same block size but an increasingly digest size. Because they inherit the structure of SHA-1, that has been broken and should never be used anymore, it can be possible that soon they will not be secure enough anymore, so SHA-3 has been already developed and should be the choice to adopt once its strength will be proven. SHA-2 family was a quick fix after the SHA-1 attack, and it was developed by making the digest longer. Main algorithms are: SHA-256 that uses a 32-bit word and SHA-512 that uses 64-bit word (suitable for 32/64-bit PCs respectively).

The digest length is important to avoid **aliasing**, that is collision of two different message on the same digest: breaking a digest algorithm means finding a second message that will generate the same digest of the first one. If the algorithm is well designed and generates a digest of N bits, then the probability of aliasing is $\boldsymbol{P_A \propto \frac{1}{2^{Nbits}}}$. This is the reason digests with many bits are required (because statistical events are involved).

As a consequence of statistical considerations (see "birthday paradox"), a N-bit digest algorithm is insecure when more than $2^{\frac{N}{2}}$ digests are generated because the probability to have two messages with the same digest is $P_A \sim 50\%$. If an attacker sniffs the network, he can simply store all the messages and its digests passing by. With many messages and digests stored the probability to have the same digests for two different messages is high. At this point it is possible to exchange the two messages, because they have the same digest. A cryptosystem is "balanced" when the encryption and digest algorithms have the same resistance: so, SHA-256 and SHA-512 have been designed for use respectively with AES-128 and AES-256, while SHA-1 (i.e., SHA-160) matched Skipjack-80 (an algorithm developed by UK secret service). Notice that AES have half bits of SHA (due to $\frac{N}{2}$).

SHA-3 family

Because of the already cited problem with SHA-2 family, SHA-3 was a competition for a new dedicated hash function: the winner was announced on 2 October 2012 and was Keccak (authors = G.Bertoni, J.Daemen, G. Van Assche (STM), M.Peeters (NXP)). The NIST team praised the Keccak algorithm for its many admirable qualities, including its **elegant design** and **its ability to run well on many different computing devices**. The **clarity of Keccak's construction** lends itself to **easy analysis** (during the competition all submitted algorithms were made available for public examination and criticism), and Keccak has higher **performance in hardware** implementations than SHA-2 or any of the other finalists.

Keccak is used to define various hash functions (FIPS-202): there are three versions of SHA-3, which are intended as the replacement of the corresponding hash functions in SHA-2 family, because they produce the same output length and have the same strength. Moreover, there are *variable-length output*, since it is possible to parametrize the output length: of course, shorter digest leads to weaker protection.

function	output	block	capacity	definition	strength
SHA3-224(M)	224	1152	448	Keccak[448](M || 01, 224)	112
SHA3-256(M)	256	1088	512	Keccak[512](M || 01, 256)	128
SHA3-384(M)	384	832	768	Keccak[768](M || 01, 384)	192
SHA3-512(M)	512	576	1024	Keccak[1024](M || 01, 512)	256
SHAKE128(M,d)	d	1344	256	Keccak[256](M || 1111, d)	min(d/2,128)
SHAKE256(M,d)	d	1088	512	Keccak[512](M || 1111, d)	min(d/2,256)

Figure 24 Strength of different hash functions belonging to SHA-3 family as function of digest length and block size

Other variations have been defined (NIST SP.800-185):

- cSHAKE (customizable domain parameters);
- KMAC (keyed digest);
- TupleHash (hash of a tuple, where sequence matters);
- ParallelHash (fast hash by exploiting internal CPU parallelism).

KDF (Key Derivation Function)

Cryptographic hash functions are not only for digests, but they can be used also to derive (create) a key: a cryptographic key must be random, that is 1 and 0 bits are to be uniformly distributed. However, typically users insert passwords or passphrases, that are guessable and not random. This happens because it is impractical for a user to remember a meaningless sequence of bits. Usually, a key is derived from a password or passphrase inserted by the user, so that $K = KDF(P, S, I)$, where:

- P is the user's password or passphrase;
- S is a salt, an unpredictable variation to make K difficult to guess even knowing P;
- I is the number of iterations of the base function (to slow down the computation and make life complex for attackers).

When providing information to the destination, the passphrase is transmitted in a secure way, while the salt will be inserted in the message itself: the salt is something like an IV, a large random number used to prevent precomputations.

There two main key derivation functions which are based upon cryptographic hash functions:

- **PBKDF2** (*RFC-2898*) uses SHA-1, with $|S| \geq 64$ and $I \geq 1000$:
  The function is $\boldsymbol{DK = PBKDF2(PRF, PWD, Salt, C, dkLen)}$, where:
  - *PRF* = pseudorandom function of two parameters with output length hLen (e.g., a keyed HMAC);
  - *PWD* = password from which a derived key is generated;
  - *Salt* = cryptographic salt;
  - *C* = number of iterations desired;
  - *dkLen* = desired length of the derived key;
  - *DK* = generated derived key = $T_1 || T_2 || \ldots \left|\left| T_{\frac{hLen}{hLen}} \, (where\ each\ |T_i| = hLen \right)\right.$.

  An important application of this KDF is in wireless networks, specifically in *WPA2* solutions. When the passphrase is entered, there is an implicit use of this function: $DK = PBKDF2(HMAC-SHA1, PWD, ssid, 4096, 256)$, where *ssid* is the name of the wireless connection.

  However, PBKDF2 can be attacked because *C* may be large, but this requires only a lot of computation but not a lot of RAM, hence it can be attacked by ASIC or GPU. A countermeasure is to increase the RAM needed for the attack. A public competition about that was launched in 2013 and on 20 July 2015 the winner was "Argon2".
- **HKDF** (*RFC-5869*) uses HMAC.

We have seen that a digest could be used in pair with a message to achieve integrity; typically, the part added to the message for integrity is called **MIC** (**M**essage **I**ntegrity **C**ode), because of the code added to the message to protect its integrity. If a message is not being modified, it means that it is also authentic and in this case the code is also named **MAC** (**M**essage **A**uthentication **C**ode). They mean the same thing, but MIC is usually used in the telecommunication and networking environment, while MAC is mostly used in the application field. Usually when talking about MIC or MAC, we are also assuming that the digest is keyed, as we will see next. Since this code is an addition to the original message, usually also a unique identifier for the message is included to avoid *replay attacks*, and that is called **MID** (**M**essage **ID**entifier). In this context a replay attack is not an attack to the integrity of a single message but to the integrity of the transmission process.

Protecting the digest

The problem now to be faced is how to protect the digest: in the simple case in which a digest is used to verify that a file on disk has not been changed, a good approach could be to store the digest in a different and secure place, so that it will not be altered. When sending data, the digest cannot be exchanged out of band, so it means that the digest must be intrinsically secure. There are two mostly adopted techniques:

- Authentication by means of **keyed digest**;
- Authenticated encryption.

Authentication by means of keyed digest

The digest is sent along with the message, but it is calculated not only on the data but also on a secret key, that is shared between sender and receiver. So, these operations are performed:

- **Sender**: $d = digest(K, M)$;
- **Transmission**: M || d;
- **Receiver**: $d' = digest(K, M)$;
- **Verification**: if ($d == d'$) then OK else it means that something has been changed during transmission (the data or the digest, it does not matter).

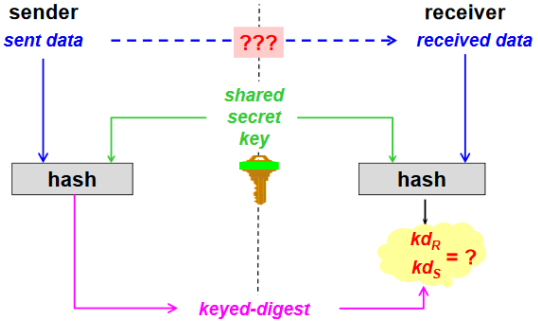


Figure 25 A scheme of how a keyed digest works

Advantages of this technique are that there is only one operation (calculation of digest) and that only few data are added. It is worth noticing that this mechanism guarantees authentication, since only who knows the key can compare the transmitted digest with the digest calculated on the received data. This is the fastest solution to provide integrity and authentication to the data. However, it does not provide **non-repudiation**: let us suppose that the receiver of a commercial order wants to take to the court the sender of that order because he repudiates the order itself, then the receiver would show as a proof the message along with the digest that proves integrity and authenticity. In this case is not possible to say that the sender originated the order, since the receiver has all the means for creating it by himself: he has the data and the key upon which compute a valid keyed digest.

So, this solution guarantees authentication since the receiver can be sure that the data is coming from the sender and it has not been modified, but it does not provide non-repudiation by itself since it is impossible for one of the two parties to demonstrate to a third party that the other created the message.

Keyed digest: possible mistakes

If $kd = H(K|M)$ (which means that you pre-append the key at the message) then it is possible to change the message adding at its end one or more blocks, such that $kd' = H(K|M|M') = f(kd, M')$: this means that modifying the data by adding one more block is sufficient to calculate a new keyed-digest without knowing the key.

If $kd = H(M|K)$ then it is possible to change the message adding before it a suitable block, such that $kd = H(M'|M|K) choosing\ M's.t.\ IV = f(IV, M')$.

Protection countermeasures are:

- insert in the digested data also the length of M;
- define $kd = H(K|M|K)$;
- use a **standard keyed digest**: the best keyed digest widely used nowadays is HMAC.

Keyed digest:

HMAC is defined in RFC-2104 and it use a base hash function H. So, HMAC is not a hash function but it is a way to compute a secure MAC starting from a hash function H. The base hash function H must have a block size B, with an output L byte long, with $B > L$. Then there are some definitions:

- **ipad** = 0x36 repeated B times; *it is a padding composed by 0x36 repeated has many times to fill all the block.*
- **opad** = 0x5C repeated B times;
- **deprecated keys** s.t. $|K| < L$; *never use a key smaller than the output*. For example, if HMAC with SHA-1 is being used, it requires a minimum of 160-bit key.
- if $|K| > B$ then $K' = H(K)$ else $K' = K$; if the key is bigger than the block, then the size of the key is reduced by computing the hash of the key (otherwise the original one is used).
- if $|K'| < B$ then K' is 0-padded up to B bytes;

Given that, $\mathbf{hmac} = \mathbf{H}(\mathbf{K'} \oplus \mathbf{opad} \,|\mathbf{H}(\mathbf{K'} \oplus \mathbf{ipad} \,|\mathbf{data}))$.

The K' is XORed with ipad and it is pre-appended to the data. Then, the hash is computed. After the hashing, the K' is XORed with opad and it is pre-appended to the result of the hash and the hash is again computed. It is easy to see that basically *HMAC* is a double hash of the data composed someway with the key. This is much stronger than just pre- or post-appending the key to the data. This is the most widely solution used when is needed to create a MAC (or MIC, it is the same). HMAC is typically used to protect integrity. If confidentiality must also be protected, there is CBC-MAC.

CBC-MAC

Sometimes we do not need only integrity, but also confidentiality, so computing encryption. In that case, rather than using another function (for example we are using AES for encryption and we need also to have SHA-256 for integrity, that is we need to have more code, that is not always possible for example in embedded systems), it is possible to compute a MAC not based on a hash function but based on a symmetric algorithm: this is named **CBC-MAC**.

It exploits a block-oriented symmetric encryption algorithm, in CBC mode with a null IV, taking as MAC the last encrypted block. The message M is split in N blocks $M_1 \ldots M_N$, then taking $V_0 = 0$ for $(k = 1 \ldots N)$ do $V_k = enc(K, M_k \oplus V_{k-1})$; finally, CBC-MAC = V_N. So, the size of the CBC-MAC is equal to the size of one block of the underlying symmetric encryption algorithm. If for example we are using AES-CBC-MAC, then the digest will 128 bit long. The best version of DES-based CBC-MAC is the *Data Authentication Algorithm* (standard FIPS 113, ANSI X9.17).

CBC-MAC is secure only for fixed-length messages, because otherwise attacks that extend the length of the message can be successful; for variable-length messages is better to use **CMAC**.

Integrity and secrecy: how to combine?

We discussed that in most applications we need both integrity and confidentiality. This can be achieved through two orthogonal operations: symmetric encryption with a key K_1 for **secrecy** and key digest (MAC) with K_2 for **integrity**. There are various ways to combine these operations:

1. **Authenticate-and-encrypt** (**A&E**): first the plaintext is encrypted with K_1, then the result is concatenated with the MAC computed on the plaintext with K_2: in formulas, $enc(K_1, p)|mac(K_2, p)$. In this way nobody can read the plaintext because it is encrypted, and nobody can alter the cyphertext because after decryption it is possible to compute again the MAC and any difference will be noticed.
   The main drawback is that for verifying integrity the message must be decrypted first (an operation that needs time), because the MAC in computed on the plaintext, so there is a vulnerability against DoS attack. Moreover, for the same reason, the MAC may leak some information about the plaintext. It is not the best solution, but it is used by the SSH protocol.
2. **Authenticate-then-encrypt** (**AtE**): first the MAC is computed with K_2 on the plaintext, then it is concatenated with the plaintext and the result encrypted with K_1: in formulas, $enc(K_1, p\,|mac(K_2, p))$. For reasons like the previous case, this solution is still vulnerable to DoS attacks since it is necessary to decrypt the whole message before being able to verify integrity by

computing the MAC. However, this solution is better than the previous one because it does not leak information, since the MAC is not sent in clear, but it is encrypted.
This solution is used by SSL and TLS up to version 1.2: TLS version 1.3 uses a completely different approach.

3. **Encrypt-then-authenticate** (**EtA**): first the plaintext is encrypted with K_1, then the result is concatenated with the MAC computed on the ciphertext obtained in the previous step: in formulas, $enc(K_1, p)|mac(K_2, enc(K_1, p))$. In this way it is possible to avoid DoS attacks, in fact an alteration on the ciphertext will be immediately noticed, that is without first decrypting the message. This is the solution used by IPsec.

Security of these compositions

Important to be noticed: improper combination of secure algorithms may lead to an insecure result!

- **Authenticate-and-encrypt** (**A&E**): insecure unless performed in a single step;
- **Authenticate-then-encrypt** (**AtE**): secure only with CBC or stream encryption;
- **Encrypt-then-authenticate** (**EtA**): the most secure mode but beware of implementation errors: for example, one should always include IV and the name of the algorithms in the computation of the MAC, otherwise some attacks are possible.

Because neither of these three solutions are perfect, current efforts are towards a joint AE algorithm: this is named **authenticated encryption**.

Authenticated encryption

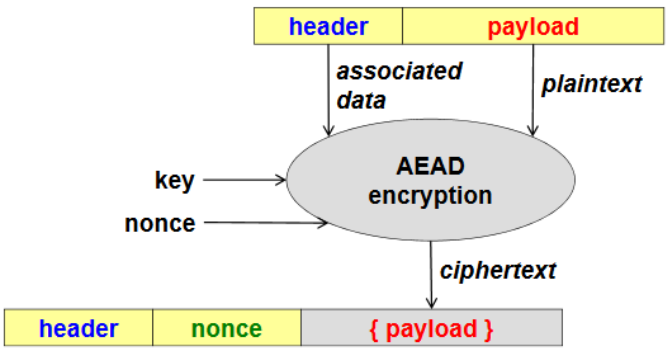
The idea is to use one single operation for privacy and authentication (and integrity). This will have the following benefits:

- just one key and one algorithm;
- higher speed;
- less error likelihood in combining the two functions. This last aspect is also sometimes requested by applications, that could have different requirements in term of integrity and privacy. Let us analyse two examples:
    - *network packets*: it is not possible to encrypt the whole packet, in fact the IP header is needed for forwarding it through the network, so the payload needs privacy and must be encrypted, and the whole packet needs integrity;
    - *electronic mail*: similar reasonings hold also for this case. In fact, the body can be encrypted, but for example the "To:" field cannot, otherwise the mail server will not know what the destination is so where the message should send the message to.

Moreover, there are other motivations for such an algorithm, because the normal encryption modes are subject to chosen-ciphertext attacks when used on-line:

- the attacker modifies a ciphertext;
- then observes if the receiver signals an error or not (e.g., **padding oracle** or **decryption oracle** attack).

There are various ways for performing authenticated encryption, one of the most important is *Authenticated Encryption with Associated Data* (**AEAD**), documented in RFC 5116.



This is exactly the kind of solution needed for e-mail and network packets: in principle authenticated encryption is an algorithm that provides both two functionalities cited above on a plaintext. AEAD on the contrary makes a

distinction between the part that needs privacy and the part that needs integrity. Referring to the above figure, the header is what that needs only integrity and authentication (for this reason also called *associated data*), while the payload is the part for which confidentiality is also needed (for this reason referred as *plaintext*). This algorithm needs a single symmetric key and a nonce to avoid also replay attacks: the result is the ciphertext, that is placed inside the packet as encrypted payload. In front of that, in clear there are the header and the nonce: inside the encrypted payload there is one part that provides integrity not only for the payload itself but also for the header and the nonce, normally this part is usually called **tag**.

IGE (Infinite Garble Extension)

It is an authenticated encryption algorithm, so it provides all in one confidentiality, integrity, and authentication. Basically, it is like the CBC with one addition: in CBC, a modification in one block of the ciphertext affect only the block itself and the next one (and then propagation stops). On the contrary, IGE uses a mechanism by which a modification in any block will affect every block after the mangled one.

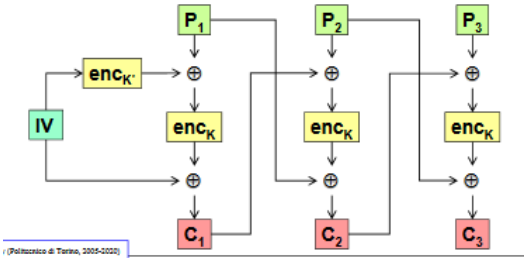


Figure 26 How Infinite Garble Extension (IGE) works: every block after the mangled one will be affected

This is important because it means that by putting a last block of plaintext with fixed content (e.g., composed by all zeros or containing the number of blocks of the plaintext), in the decryption phase it is sufficient to look at the last block and check if it contains the expected value: if not, one can say immediately that there has been an attack against integrity. In this context, the last block is what previously was called **tag**.

However, this algorithm is not recommended, it is not very good, but the professor explained it for teaching purposes.

Authenticated encryption: standards

ISO/IEC 19772:2009 defines 6 standard **modes** (applying these modes with basic encryption algorithms it is possible to get authenticated encryption with associated data):

- **OCB 2.0** (*Offset Codebook Mode*) [single-pass AEAD, patented];
- **AESKW** (*AES Key Wrap*);
- **CCM** (*CTR mode with CBC-MAC*) [double pass];
- **EAX** (*Encrypt then Authenticate then X(trans)late*) = CTR + OMAC [double-pass AEAD, free];
- **Encrypt-then-MAC**;
- **GCM** (*Galois/Counter Mode*).

GCM as an example of AEAD

An algorithm applied in GCM mode encryption takes these parameters:

- On encryption, $(\boldsymbol{C},\boldsymbol{T}) = \boldsymbol{algo}_{\boldsymbol{GCM}_{\boldsymbol{enc}}(\boldsymbol{K},\boldsymbol{IV},\boldsymbol{P},\boldsymbol{A})}$, with:
  - IV size [1 ... 2^{64} bits] (96 bits is most efficient);
  - P of size [0 ... 2^{39}– 256 bits];
  - A (associated authenticated data) of size [0 ... 2^{64}bits];
  - C has the same size as P;
  - T is the authentication tag with size [0 ... 128 bits]: comparing with cryptographic hash function, this tag is not very large, and we argued that a big integrity code is necessary for strong security. Beware however that since here we are not using a hash algorithm, the security is equal to the number of bits, while for cryptographic hash functions we saw that the protection is half of the length of the digest. So, in this case a tag of 128 bits gives the same protection of a digest long 256 bits.
- On decryption, $\boldsymbol{P} = \boldsymbol{algo}_{\boldsymbol{GCM}_{\boldsymbol{dec}}(\boldsymbol{K},\boldsymbol{IV},\boldsymbol{C},\boldsymbol{A},\boldsymbol{T})}$, where: P is the original plaintext (if authentication is OK), or a special failure value if the authentication checks failed.

GCM is defined for only the encryption algorithms with 128-bit block (for example we cannot have DES in GCM, but it is possible to have AES).

CCM as an example of AEAD

It is **Counter-mode with CBC-MAC**: first an authentication tag of (plaintext + associated data) is computed by CBC-MAC, then the plaintext and the tag are (separately) encrypted in CTR mode: so, it is a double pass algorithm. It is defined for encryption algorithms with 128-bit block.

Authenticated encryption: applications

- TLS-1.3 uses GCM and CCM;
- 802.11i uses CCM;
- ZigBee (short range protocol for IoT) uses CCM* (=CCM + auth-only + enc-only);
- ANSI C12.22 (network transmission of electronic measures, e.g. house power meter) uses **EAX'** (famous case of **failure**): nowadays it is common to use electric wire to send electronic measures of power meter. Because of there is no firewall over an electric wire, by attaching a proper device to the electric network, it could be possible to read measure of other people or create fake transmissions, so protection is needed for privacy (for example a house not consuming could signal that nobody is at home and so it can be robbed) and for authenticity (the measures sent must be the real ones).
    - The modification of the base algorithm created a vulnerability, so the algorithm has been proven to be broken (article by Minematsu, Morita and Iwata);
    - EAX had a formal proof of its security ... but ANSI sacrificed it for 3-5 less encryption steps and 40 bytes less memory usage (so important for embedded systems?).

Comparison of AE modes

- **GCM**: *the most popular, on-line single-pass AEAD, parallelizable, used by TLS, present in openssl*;
    - for encryption, generates ciphertext + authentication tag;
    - for decryption, first computes authentication tag and only if it matches the one in input then the ciphertext is decrypted;
    - fast on Intel architecture (~4 cycle/byte) using AES-NI for encryption and PCLMULQDQ for the tag.
- **OCB 2.0**: *the fastest one, on-line single-pass AEAD, GPL patented so scarcely used, now free but for military uses*;
- **EAX**: *on-line double-pass AEAD, slow but small (uses just the encryption block) so very good for constrained systems*;
- **CCM**: *off-line double-pass, the slowest one*;
  Note that double-pass is 2x slower than one-pass (in software).

Competition for AE

The subject of an AE algorithm is so important that there was a competition named **CAESAR** (*Competition for Authenticated Encryption: Security, Applicability, and Robustness*). The final selection was expected in December 2017 and it should have produced an algorithm portfolio, but the challenge ended without a result.

NIST recommendations for using block encryption algorithms

NIST has provided through years recommendations for block algorithms, all contained in the special document *800-38*, which is periodically updated.

- *part A (dec'01)* = five confidentiality modes (ECB, CBC, CFB, OFB, and CTR);
- *addendum (oct'10)* = three CBC ciphertext stealing;
- *part B (may'05)* = one authentication mode added called CMAC (more less like OMAC, that is better than XCBC, that is better than CBC-MAC);
- *part C (jul'07)* = added auth. encryption with CCM;
- *part D (nov'07)* = high-speed auth. enc. with GCM;
- *part E (jan'10)* = confidentiality for block memories = XTS-AES;

- draft = use of AESKW for auth. enc. of keys.

Authentication by digest and asymmetric cryptography: digital signature

As we have seen, authenticated encryption provides integrity, authentication, and confidentiality to data but, as we discussed keyed digest, it does not provide non-repudiation. As seen, this happens because the peers share the key upon which compute a valid keyed digest, so it is not possible to say which of the two peers has computed it.

Non-repudiation is achieved by using digest and asymmetric cryptography: the digest is encrypted with the private key of the sender, so that the use of his public key will prove the source. Using formulas, such a communication takes place as following:

- **Signer**: H = $enc\left(S_{Kpri}, hash(M)\right)$
- **Transmission**: $M \mid H$
- **Verifier**: X = $dec\left(S_{Kpub}, H\right)$
- **Verification** phase: if ($X == hash(M)$) then OK else it means that something has been changed during transmission.

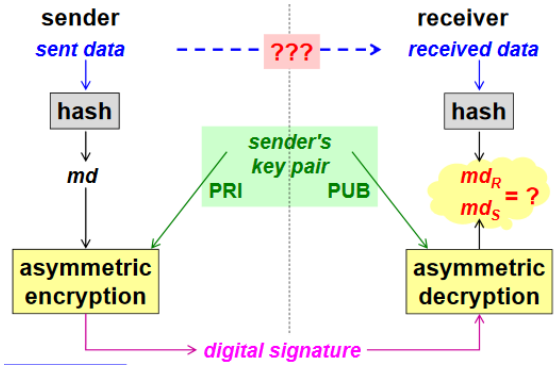


Figure 27 Digital signature: computing and verifying a message digest encrypted via asymmetric encryption

This is the mechanism by which **digital signatures** are implemented. The last piece to obtain full non-repudiation is a mechanism by which must be possible to guarantee that the association between a public key and an identity is true: this is called **public key certificate**.

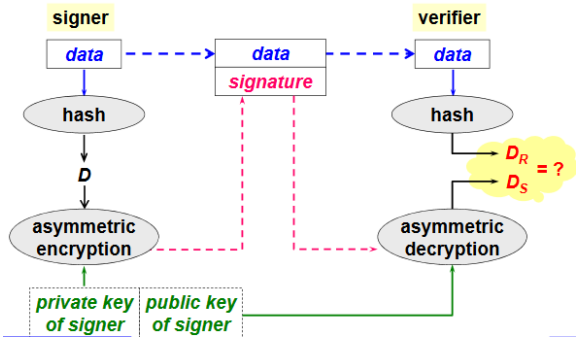


Figure 28 Signature creation and verification

An attack that it is possible to perform against this schema is *modification of digital signature*: changing any bit of the encrypted data leads to failure in verification phase, but also does any modification to the digital signature: no matter if the data is correct, it is not possible to say whether the modification affected the data or the signature.

Another case worth discussing is taking a digital signature performed by someone else, then taking that person's public key as well (to make sure the public key fits the digital signature), and then attaching it to a another file, so that it is attributed to that person: this will still result in a failure in the verification phase, since the digital signature contains the encrypted digest of the original data: the digital signature is bound to the data, so it is not possible to *attach a signature to other data*.

Authentication and integrity: analysis

We have seen two ways to obtain **authentication** and **integrity**:

- by means of a **shared secret**:
  - useful only for the receiver;
  - cannot be used as a proof without disclosing the secret key;
  - not useful for non-repudiation.
  - Finally: good for two peers who trust each other, as a protection against a third party;
- by means of **asymmetric encryption**:
  - being slow it is applied to the digest only;
  - can be used as a formal proof;
  - can be used for non-repudiation, providing that it is possible to demonstrate who is the owner of the public key (**public key certificate**);

Digital vs. handwritten signature

Digital signature provides authentication and integrity, while *handwritten signature* provides authentication only. Thus, the digital signature is better because it is tightly bound to the data. Please note: each user does not have a digital signature but a private key, which can be used to generate an infinite number of digital signatures (one for each different document).

Public key certificate

As we have seen so far, a digital signature has a great benefit, that is it can provide non-repudiation. However, non-repudiation is achieved if, in addition to perform a correct digital signature, it is possible to ascribe with certainty the public key to an actor's private key. The attribution of a key to an actor is usually done through a **public key certificate**: it is "*a data structure used to securely bind a public key to some attributes*". Typically, it binds a key to an identity, but other associations are possible too (e.g., IP address). The security in this binding is provided by a digital signature: the entity that created the certificate, named **certification authority** (CA), digitally signs the certificate, so that it is at the same time:

- a proof of *authentication*: it is a valid certificate because it has been created by a certification authority;
- a proof of *integrity*: the data are correct, and they have not been modified since the document was issued.

Like a common identity document, also a public key certificate has a **limited lifetime**. It can also **be revoked** on request both by the user and the issuer.

Formats for public key certificates

- **X.509**:
  - v1, v2 (defined by ISO): not very successful;
  - **v3** (ISO + IETF): has achieved a great usage, it is nowadays the most common format, because it can be applied to the security of several internet protocols;
- **non-X.509**: since X.509 was developed by ISO, which is usually seen as the standardization body for big corporations, there are some people who do not trust big corporations and made an effort in creating alternative certification structures, such as:
  - **PGP:** has been used (and still used) mostly in the underground movement;
  - **SPKI** (IETF)

  None of these have achieved great diffusion.
- **PKCS#6**: it is a standard that was promoted by the RSA before X.509 was invented, so it is partly compatible with X.509, but since then RSA itself declared it obsolete.

Structure of a X.509 certificate

A X.509 certificate is composed by:

- **Version**: since there are more versions of the same format as seen previously;
- **Serial number**: each certificate is uniquely identified;
- **Signature algorithm**: since the certificate is protected with a digital signature, in its format there is the information about the algorithm used to perform the signature;
- **Issuer**: is the name of the entity that created the certificate, the certification authority. The syntax used to specify this information is called **DN** (distinguished name), that is composed by three different fields:
  - **C** stands for country (e.g., Italy);
  - **O** stands for organization (e.g., Politecnico di Torino);

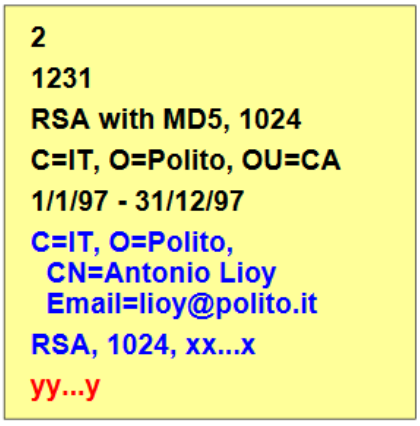


Figure 29 Example of a X.509 public key certificate structure

    - **OU** stands for organizational unit, which means that this is the department who created this certificate (e.g., certification authority).
- **Validity**: defines the period for which the certificate is considered valid. Out of that period, the CA does not guarantee the correspondence.
- **Subject**: the entity that is controlling the private key corresponding to the public key being certified. Also, here it is used the DM to identify that entity, with other fields not mentioned earlier:
    - **CN** stands for common name (e.g., Antonio Lioy);
    - **Email**, that is the e-mail of the entity. It is important to include this fields because public key certificates are used to protect internet applications, for example to send an e-mail to "Antonio Lioy", the mail server needs to be able to understand what the certified mail for the entity is (so, the e-mail field is required). The same reasoning applies to other applications for which protection is wanted.
- **subjectPublicKeyInfo**: is the field containing the public key; it specifies the algorithm, the length of the key and all the bits that constitute it;
- **CA digital signature**: it is the digital signature of the certification authority computed over the certificate; in this way any modification to it can be detected, as seen previously.

So, this mechanism finally solves the problem of knowing who the actor controlling the private key corresponding to the public key is.

PKI (Public-Key Infrastructure)

This is an infrastructure which is performing two kind of tasks: *technical* (create certificates) and *administrative* (before creating a certificate, checking the identity of the requestor is needed). So, this infrastructure oversees *creating, distributing,* and *revoking* the public key certificates.

Certificate revocation

Every certificate has a certain validity, but it can be revoked before its expiration date. The revocation can be requested by the owner or by the creator of the certificate:

- The owner can request the revocation in case he has no more control on the private key (e.g., private key stored on a stolen smartcard). This case looks like when a credit card is lost: the owner should go asking to revoke the certificate as soon as possible, because if the thief uses that private key, it will appear as if the original owner has performed the signature. This is the most common case in which a certificate is revoked;
- The creator can revoke the certificate in case a certificate has been created for a fake identity: in fact, if someone has managed to demonstrate a fake identity to a certification authority, he will receive a valid certificate. Nevertheless, if later the certification authority discovers the deception, it will revoke the certificate automatically.

When a certificate is revoked, the certification authority creates a list which says that the certificate is no more valid. The certification authority does not know when and where the stolen key will be used, so when a signature is used to protect some transactions, the transaction is received by some actors (the receiver) that must check that the certificate was valid at signature time. This actor (which receive the signature) and has the task to verify if the signature was valid or not is named **relying party (RP)**.

This means that before trusting the certificate, it should be checked if it is valid or not.

Revocation mechanisms

There are two mechanisms to perform a check on the signature:

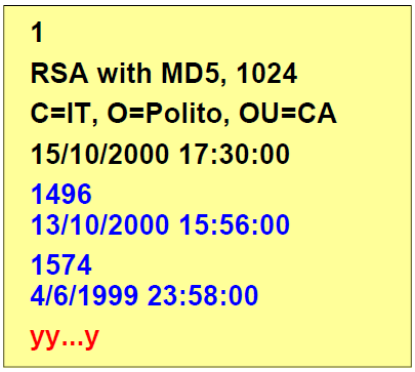
- **CRL (Certificate Revocation List):** the certification authority creates a list of revoked certificates. The list is digitally signed by the CA, because otherwise someone could cancel a certificate from the list or vice versa a certificate could be added to it to "block" someone: the CA provides authenticity and integrity for this list by digitally signing it. Since it is a list with **all** revoked certificates, this will

permit the receiver to verify the validity since the certificate was issued. For example, if today a document is received and it was signed 3 months ago, we must check if the key was valid 3 months ago. If the certificate has been removed today, it does not affect the received document, because the check must be done against the time of the sign. The problem is that we must download the full list only to check one single certificate, but this is the only way to know the history of a certificate.

- **OCSP (On-line Certificate Status Protocol):** this mechanism is preferred for real-time checks, for example for transactions. In this case, we need to check if the certificate is valid **now** and the service needing the check do not care previous states of the certificate. OCSP is a client-server protocol in which is possible to request to a specific service if the certificate is valid or not at the current time. The response is signed by the server, so that is not possible for someone to provide a fake response.

Structure of a X.509 CRL

The X.509 CRL has got the same parameters as the X.509 certificate, such as: *version, signature algorithm and issuer*. But the X.509 CRL structure has got the "**thisUpdate**" parameter, which is the time of when the CRL was created. This parameter is important because if a check is performed on a signature that was made before this date, the information is contained in the CRL. But if the check must be done on something happened after the *thisUpdate*'s date, then this CRL is not good for our purposes and we need to get a fresher CRL. After this parameter, there is a list of the revoked certificates: for each certificate is reported the *userCertificate* and the *revocationDate*.



Verification of a signature/certificate

Suppose we want to verify a signature: we will perform its verification by requesting the public key certificate, that is signed by the entity that provides it (e.g., the certificate of "Antonio Lioy" is signed by "Politecnico di Torino"). A question now arises: how it is possible to verify that this last signature is valid? To check if the signature of the CA is valid, a public key certificate of that CA is needed, which is created and signed by another CA, whose signature in turn needs to be verified. It is easy to see that the problem is recursive: it becomes thus necessary to have a hierarchical infrastructure for certification and distribution of public-key certificates.

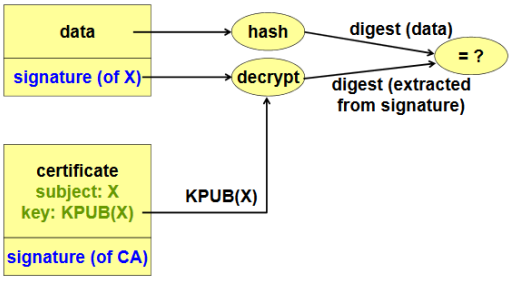


Figure 30 The problem that arises when verifying a signature over data sent by an actor. To check the signature over "data", the public key certificate for X is needed, and it is provided by a CA. In turn, that certificate needs the same treatment: the signature of the CA needs validation.

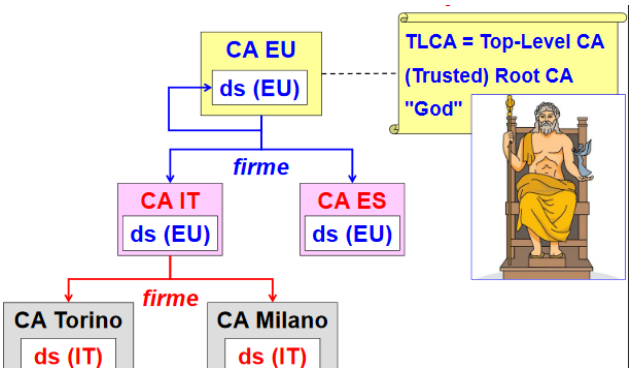


Figure 31 The hierarchical infrastructure for certification and distribution of public-key certificates.

Such hierarchical infrastructure is depicted on the left. For example, to check the signature of "Politecnico di Torino", that created the public key certificate for "Antonio Lioy", the certificate of "CA Torino" is needed, which contains the signature of the "CA IT"; to check if this one is good, the certificate of the entity that signed the "CA IT" certificate is needed, so the "CA EU" must be get. This last CA ends the chain, in fact the "CA EU" is signed by the European CA itself: this last certificate is trusted a priori (*self-signed*), it cannot be verified, and for this reason is also called **TLCA**, **Root CA** or even **God**.

Please note that in the world there is not only one TLCA, neither there are a few: typically, in computer system there are quite a lot of trusted CA, and this is due to two main reasons:

1. **Security**: the one who would control the only TLCA would also control the other ones, by faking them;
2. **Commercial**: to obtain a certificate by a certification authority, a price must be paid to it.

For the second reason, the list of trusted CA can be different on systems produced by different manufacturers: since it is a matter of on who place the trust, one system can accept or not a CA as a trusted one. For example, in the past Microsoft performed only light checks on incoming trusted CA but requested the payment of 250.000 $ to be in its list.

The problem that arises is: can't an attacker fake the whole tree? Of course, that is possible: if the device is left unattended and the attacker manages to have administrative privileges, it is possible to add another (fake) Root CA, making the whole hierarchical system useless for that device. The list of Root CA can be managed either by the operating system or by the application being used.

Performance

Cryptographic performance does not depend on RAM but on CPU (architecture and instruction set) and cache size: some CPUs have special instructions for cryptographic algorithms. In general, performance is not a problem on clients (except when not very powerful, for example in case of IoT, embedded system, or client overloaded by other applications). Performance can become a problem on servers or when implementing security on the network nodes (e.g., router); in these cases, it is possible to use:

- *cryptographic accelerators* (**HSM, Hardware Security Module**): it has some primitive instructions to perform some security algorithms;
- **special-purpose accelerators**, that is accelerators that not only implement in hardware cryptographic operations but also the packet processing for some security protocols (e.g., *SSL*, *IPsec*), or **generic accelerators**, which is for example just speeding up AES computation for whatever protocol is wanted to implement.

The following figures show examples of performance: it is possible to notice that for computing cryptographic hash function, the performances are much better if the operation involves packets whose size is higher. This is no longer true when talking about encryption algorithms, but it is possible to notice that, obviously, 3DES is three times slower than normal DES, that AES is faster than DES even if it guarantees much better security, and that RC4 is much faster than the others because it is a stream algorithm.

	[64 B/packet]	[1024 B/packet]
hmac(md5)	193.8 MB/s	630.5 MB/s
des-cbc	90.5 MB/s	91.0 MB/s
des-ede3-cbc	33.7 MB/s	33.9 MB/s
aes-128-cbc	171.3 MB/s	178.4 MB/s
rc4	783.1 MB/s	637.9 MB/s

Figure 32 Performance of crypto hash functions and encryption algorithms on a powerful CPU (Intel 7-7600U @ 2.80GHz)

	[64 B/packet]	[1024 B/packet]
hmac(md5)	19.2 MB/s	83.6 MB/s
des cbc	14.4 MB/s	14.5 MB/s
des ede3	5.2 MB/s	5.2 MB/s
aes-128	15.6 MB/s	15.9 MB/s
rc4	80.9 MB/s	86.4 MB/s

Figure 34 Performance of crypto hash functions and encryption algorithms on a CPU employed in embedded systems (Intel P3 @ 800 MHz)

Considering asymmetric cryptography, performances is usually not expressed in MB/s but by the number of signatures created or validated in one second. It is possible to notice that these algorithms are normally biased towards signature verification, which is faster: the reason behind that is that normally a signature is created once but maybe needs to be verified many times. Also notice that doubling the size of the key does not halve the performance, which degrades by a factor of ten: increasing the key length in asymmetric cryptography leads to exponential performance degradation.

rsa 1024	12093.7 firme/s	192952.5 verifiche/s
rsa 2048	1801.4 firme/s	64765.0 verifiche/s

Figure 33 Performance of asymmetric encryption algorithms on a powerful CPU (7-7600U @ 2.80GHz)

rsa 1024	94.8 signs/s	1682.0 verifies/s

Figure 35 Performance of asymmetric encryption algorithms on a CPU employed in embedded systems (Intel P3 @ 800 MHz)

In embedded systems usually old CPUs are used, and this can lead sometimes to very poor performance compared to a modern one. So, in general, before taking the decision about which algorithm to use, one should make a real-time test of that

algorithm on the specific hardware platform that will be used for the target application.

Suggested solutions for security

NSA suite B (2005-2017) and CNSA (2018)

We have seen several algorithms, several modes of application and indeed some guidelines on what to use exist. One of this is the one by **NSA** (National Security Agency of United States) has recommended **suite B** during the period 2005-2017: the name "**suite B**" comes from the fact that "*suite A*" is the set of algorithms that only the NSA uses and will not disclose. So, suite B is what NSA suggests to USA government to use when buying COTS (Commercial Off-The-Shelf) products that treat SBU (Sensitive But Unclassified) and Classified information. It contains the following algorithms:

- (**symmetric encryption**) *AES-128* e *AES-256*;
- (**hash**) *SHA-256* e *SHA-384*;
- (**key agreement**) *ECDH* e *ECMQV*;
- (**digital signature**) *ECDSA*.

For information up to Secret level it suggests AES-128 + SHA-256 + EC P-256; for information at Top Secret level it suggests AES-256 + SHA-384 + EC P-384: the algorithms are the same but the key or the digest is longer.

In 2018 NSA released a new suite named **CNSA** (Commercial National Security Algorithm Suite). It includes the following algorithms:

- (**symmetric encryption**) AES-256: now any key shorter than 256 bits is considered not enough;
    - To protect real time (e.g., stream) data, it suggests mode CTR for low bandwidth applications or GCM for ones that require high bandwidth. Moreover, GCM also provides authenticated encryption, that is an additional advantage;
- (**hash**) SHA-384;
- (**key agreement**) ECDH e ECMQV;
- (**digital signature**) ECDSA, with EC on the curve P-384.

For legacy systems, for example old systems still working that cannot perform EC or GCM, it is suggested to use:

- (**key agreement**) DH-3072;
- (**key exchange**, **digital signature**) RSA-3072;
- new quantum-resistant algorithms expected by 2022.

Length of keys and digest (NIST, 2011)

NIST in 2011 released a recommendation which gives the suggested length of keys depending on the number of years the protection should hold. In the following table:

- **FFC** stands for **F**inite **F**ield **C**ryptography (e.g., DSA, DH);
- **IFC** stands for **I**nteger **F**actorization **C**ryptography (e.g., RSA).

symm.	FFC	IFC	ECC	hash	years
80	1024	1024	160	160	< 2010
112	2048	2048	224	224	< 2030
128	3072	3072	256	256	> 2030
192	7680	7680	384	384	>> 2030
256	15360	15360	512	512	>>> 2030

Figure 36 Relation between key length and protection needs in terms of years for different type of algorithms

It is possible to notice that the key length must be very long if the data encrypted now need to remain such for years after 2030: the requirement on keys for the last two lines makes the algorithm completely unfeasible given the current CPUs' speed.

Security issues in imported products: a retrospective

Nowadays there are some reason to doubt about products, that somehow contains security mechanism, provided by other nations: for example, there is a discussion on whether using or not the 5G equipment

produces by Huawei, because there is a fear that Huawei could put malware or spy software inside their product. This situation is not new, in fact there is a precedent of similar issue.

From 1990 in USA the export of cryptographic material was subject to the same restrictions as nuclear material, unless the protection level is very low: American companies willing to export their software containing security solutions were forced to lower the protection, restricting the length of the symmetric key to 40 bits and asymmetric key to 512 bits (even in that days, computation to break such a protection would have required only few CPU hours). Examples of such limitations on software are the case of the exported versions only of Netscape and Internet Explorer: these software were hampered, so they had a reduced strength compared to the US version.

This prompted some companies around the world to develop their own solutions, so that American companies were losing market: after protesting against the government, in December 1996 US Government authorized the export of semi-robust (56 bits) cryptographic products if they incorporate **key-escrow** functions. **Key escrow** is the possibility to recover a key even without the consent of the owner. Most companied refused to implement this solution but there was one notable case of the famous software *Lotus Notes 4.x*, that used 64 bits symmetric keys, but 24 bits of them were encrypted with the NSA public-key: in this way NSA could easily retrieve these 24 bits and calculate the remaining 40 bits, while having the standard protection for anyone else. The problem was that there was no statement about who decides when it is necessary to recover a key.

The many edition of Lotus Notes

"Aside from encryption process time, U.S. government export laws limit encryption key length. These laws are the driving force behind the three major editions of Notes: North American, International, and French. Despite the different names, the product functionality is exactly the same. The difference, however, lies in the length of the keys used for encryption. The North American edition uses encryption keys that are 64-bits long. The U.S. Government, for reasons of national security, limits the length of encryption keys for export to 40 bits. To comply with these restrictions, we have the International edition.

When we generate a 64-bit key for the International edition, the top 24 bits are encrypted using the U.S. Government's public key and stored in what is called the Work factor Reduction Field (WRF). Splitting the key in this manner results in a key that is 40 bits for the U.S. Government and 64 bits for everyone else. This approach maintains a high level of security worldwide without violating the export laws of the U.S. Government.

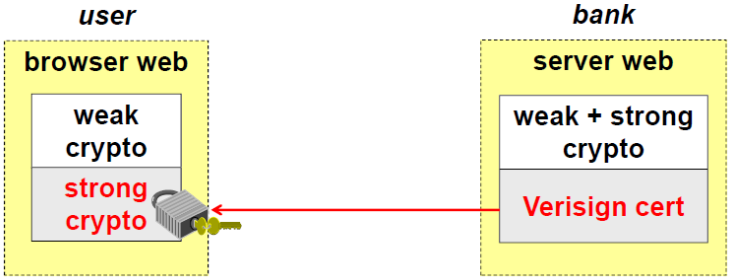
Most countries are content with the way the International edition complies with U.S. encryption key export laws. The government of France, however, found the International edition unacceptable. To comply with French law, we created the French edition, which uses a plain 40-bit encryption key and can therefore be "broken" by attackers willing to apply considerable computing power (presumably, including the French government)".

Changes in the USA cryptographic export regulations

Cases like that continued to foster the creation of other solutions, so the USA regulations changed:

- **June 1997**: permission to export secure web client and server web was granted only if used by foreign branches of USA companies or in financial environment (transactions). To verify the real use, special certificates issued by Verisign must be used, using a mechanism called **step-up** or **gated cryptography**;
- **September 1998**: permission extended to insurance and health institutions and no permission for keys up to 56 bits (because at that time was clear that 56 bits are not enough).

Step-up (gated) cryptography (dec'98)



Let us suppose that we are a bank (right side of the picture) that can buy US software and the web server will contain strong crypto, but user is Italian with an Italian Browser that has only weak crypto. When both must communicate together, the only solution is to use the weaker cryptography available. For this reason, the banks asked to government how to make it possible to communicate and here is the trick: the international version of the program contain also the *strong crypto* part, which is locked and cannot be used. But, if a bank buys a software, can also buy a *public certificate* from Verisign (the only trusted by US Government as a CA) which can perform the checks that the requestor is really a bank and not a fake one. Once bank has the Verisign cert, can send as part of the transaction also the certificate. The certificate magically opens the lock for strong crypto, but ***only*** for the communication with a bank. This is named **step-up cryptography** because you start from weakness and make one step above with stronger crypto. It is also named **gated cryptography** because the strong part is inside a gate opened only when the server submits the correct certificate.

Key recovery: another novelty?

Later the government introduced another rule: it is possible to export products with strong cryptography (e.g., 128 bits symmetric keys) if the **key-recovery functions** are incorporated. The key-recover means that if something is encrypted, and then the key is lost, it is possible to have a key-recovery to still read the encrypted data. These keys, created among with the crypto, must be placed in a **recovery centre** authorized by the USA government. Unfortunately, all these centres were all placed inside US military bases (which is not much trustable). In this way all the matters become political problems: but if you are a company and you have decided to protect data with encryption, it must be well known what happen if the key is lost, because if that happens there could be really huge problems.

With the new millennium (Gen '00) there was a new regulation with the permission to export off-the-shelf products that have passed a "*one-time review"* or products whose source code is freely available in Internet.

Summary

This is the common factor to the above regarding US laws and recent concerns about importing 5G hardware and software: all the governments try to spy about citizen, and it could be for good reasons, but this gives also concern about the freedom, in fact a government could spy on citizens to understand if they are against the government itself. Today, in the same way as US did in the past, there are some doubts that inside the equipment that manufacturers like Huawei produce there could be a spyware, or a malware controlled by the Chinese government.

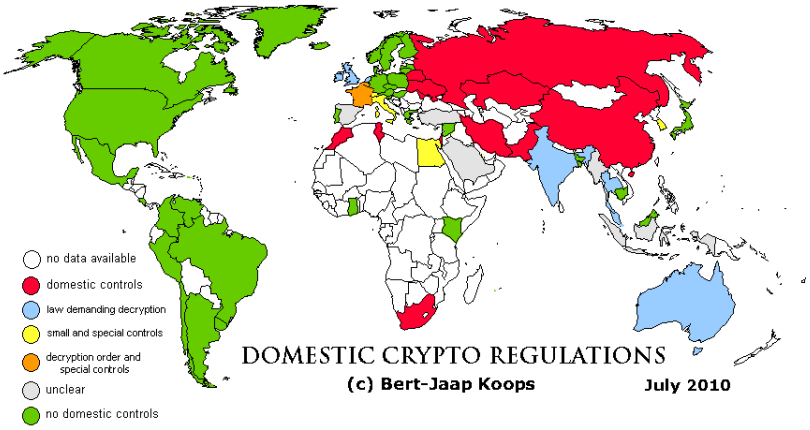


Figure 37 This is a graphic summary of the pertaining cryptography laws and regulations worldwide as outlined in the most recent version of Crypto Law Survey.

Authentication techniques and architectures

There are different definitions of *authentication*:

- RFC-4949 (**Internet security glossary**):
  - "the process of verifying a claim that a system entity or system resource has a certain attribute value";
- **whatis.com**:
  - "the process of determining whether someone or something is who or what it is declared to be";
- NIST IR 7298 (**Glossary of Key Information Security Terms**):
  - "verifying the identity of a user, process, or device, often as a prerequisite to allowing access to resources in an information system".

The important points of these definition are that they define authentication of an actor, meaning that it could be not only a human being (interacting via software running on hardware), but also a software component or a hardware element (interacting via software). Common shorthand for authentication is *authN* or *authC*, while *authZ* is used for authorization, that is different but related.

While authenticating an actor, there are three categories of **authentication factors** that can be used:

- **Knowledge**: authentication relies on something that *only the user knows*, for example static passphrase, code, personal identification number. The associated risk is in the storage, in the way it is possible to demonstrate that knowledge and in the way it is transmitted;
- **Ownership**: authentication relies something *only the user possesses* (often called an "authenticator"), for example a token, smart card, smartphone. The associated risks can be in the authenticator itself: it can be infected with a malware, or it can be manufactured in a country that imposes some government control on it, or it can be stolen, cloned, or used without the owner's authorization (e.g., forgetting a smartphone unlocked);
- **Inherence**: *something the user is*, for example a biometric characteristic (such as a fingerprint). The associated risks can be in counterfeiting and privacy: it is much worse than the previous cases, because for example a biometric characteristic cannot be replaced when "compromised". For this reason, inherence factors should be limited to very secure environments, typically it should be used only for local authentication, as a mechanism to unlock a secret or a device.

Digital authentication model (NIST SP800.63B)

In this model, an actor who wants to use a system is called an *applicant*: if it possesses an authenticator it can provide it to the **CSP** (**C**redential **S**ervice **P**rovider), or it can get one (for example, when a student is enrolled in Politecnico, he is given a smart card that works as an authenticator). The CSP is that component that will issue or enrol user credential and authenticator and verify and store associated attributes.

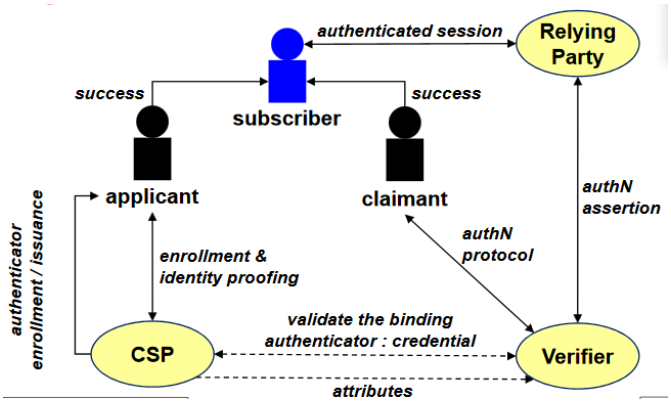


Figure 38 General model for digital authentication as described in NIST SP800.63B

When this procedure is completed successfully, the actor becomes a *subscriber*, that is an entity recorded in the authentication system. Later, when the actor wants to use some network service, typically the actor is called a *claimant*, because he claims to be a valid user: typically, an authentication protocol against a **verifier** is run, who verifies this claim. When this process end with success, the actor become a subscriber with an open authenticated session with the **relying party**, that will request and receive an authN assertion from the verifier to assess user identity (and attributes). The relying party is the end application

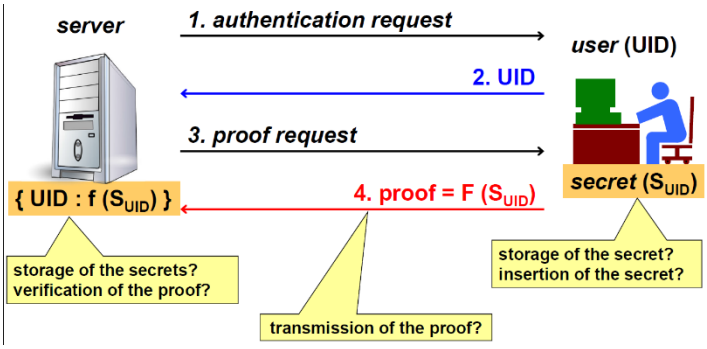
server, which requests the actor to be authenticated. The verifier may have a communication with the CSP to validate the binding between the authenticator used in the authentication protocol and the credential claimed.

Credential binds an authenticator to the subscriber, via an ID: for example, a X.509 certificate can be considered the credential, because it binds the identity and the attributes that are written inside the certificate with the authenticator, that in this case is the private key that the user controls.

These roles may be separate or collapsed together. Thinking about a Linux machine used locally: the enrolment phase is the creating of a new user with username, which is the credential, and password, which is the authenticator. In this scenario the CSP is the operating system itself and when a user wants to use a server of this machine, he needs to perform login, which is the verifier. The relying party finally is any software which is running on that machine which is using the identity as was proved by the login service of the operating system.

Another example is the use of Google Identity for different services, such as Doodle Service to agree about a date. For Doodle there is the option to use Google or Facebook Credentials. In this case the Relying Party is Doodle, while the Verifier (as well the CSP) is Google or Facebook.

User Authentication

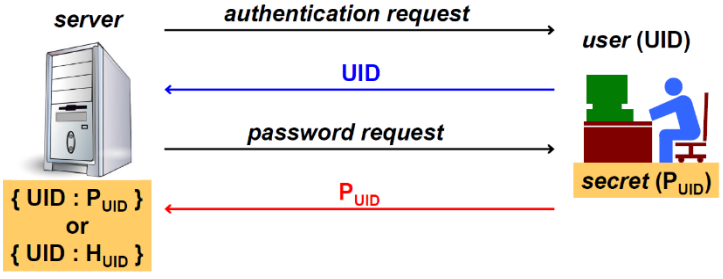


The user wants to access an application server (Relying Party). In this case the server will include both Relying Party and Verifier. The user has been identified with the User ID and has got a secret associated to that User ID. The server identifier contains a table with User ID and the result of the function f over the secret. Normally the secret should never be stored in cleartext, but of course if the function f is identity, this means you are storing the secret in cleartext, which is not recommended.

When the user wants to access the service, it receives an *authentication request*, it will provide first the *UID* and then the verifier will ask for a *proof request*, and the user will reply with the proof, which is the result of the computation with function F over the secret of the user. In this schema there are several problems to be faced: *in the user side, how is the secret stored? How is the secret provided (e.g., if it is a password inserted via keyboard a keylogger could disclose it)? Is the transmission of the proof secure? And on server side, how are the secrets related to the user stored? When a proof is received how is it verified to be the correct proof?*

Password (reusable)

Imagine that the secret is a **reusable password** (meaning that it is always the same), and the user is identified by his User ID and the secret is the password associated to that user. Server side, there is a table containing usernames and passwords in clear or a function H computed over the password. Again, there will be *authentication request*, then the user will send the *userID*, then *password request* and user response with P_{UID}.



Let us assume the network is secure and concentrate on the verifier side. The secret is the user password, and the client creates and transmits the proof typically with a function F which is the **identity function**. The proof is the password sent in cleartext, which is of course dangerous.

Looking at the server, when the server receives the password he needs to check if it is correct or not:

- *1° case*: if the function f used to store the password is the identity function, then it is the solution, because the server knows all the passwords in cleartext and verifying its correctness is simple. It is risky: if someone copies the database, will gain access to all data.
- *2° case* (suggested one): in this case f is not identity but is a **one-way hash** (that is a digest of the password) and the server does not know the password in cleartext, but just the digest. That means that the access control is a bit more complex, because when the proof is received the hash of the proof is computed and compared with the hash stored in password database. If the database is stolen, the attacker will not have a copy of cleartext passwords.

Password-based authentication is usually convenient for the user, but only if he has to remember just one password, so a reusable one. The current situation is unfortunate, because in some applications there is the need of several passwords, that cannot be kept in mind by a person, so they would need to be stored user-side, and this is source of insecurity. The **disadvantages** of password-based authentication are:

- The **user-side password storage**: it could be written on a post-it or on a client-side password manager (also called password wallet), that stores it encrypted typically using only one passphrase;
- **Guessable passwords**;
- **Server-side password storage**: the server must know the password *in cleartext* or an unprotected digest of it (dictionary attack);
- Password can be **sniffed** while it is sent across the network;
- There could be **attacks** to the password DB at the verifier (if DB contains plaintext or only obfuscated password);
- **Password guessing**: it is very dangerous if it can be done offline, for example against a list of password hashes;
- **Password duplication**: using the same password different services, due to user password reuse. This could be a problem because if the user has the same password for a high security service and for a weaker one, an attacker could discover it on the weaker system and have so the access to the high security one;
- **Cryptography ageing**: the solution adopted for verifying the secret should not be tied up to a specific cryptographic algorithm, because it could be then difficult to adapt to the need of changing the algorithm used, due to new attacks and more computing power;
- **Password capture** via server *spoofing* and *phishing*;
- **MITM attacks.**

The best practice for password is:

- You should use a **mixture** of alphabetic character (uppercase and lowercase), including digits and special characters. Unfortunately, there are a lot of systems that don't let the use of special characters or put a limit on length of password;
- Use a **long** password, at least 8 characters;
- **Never use dictionary words**, because the attacker uses dictionary from all languages;
- **Frequently change the password**: if the same password is kept for year, the attacker has more time to perform his computation. It is just needed to change password one or two times per year, to reduce the window of exposure;
- **Trying not to use passwords,** but it is unavoidable unless biometric techniques are used.

Storing password on **server-side**:

- **Never** store it in cleartext;
- If **password is encrypted**, then the server must know the key in cleartext, which can be a problem. The solution is to **store a digest** of the password but beware of the *dictionary attack* that can be made faster by a technique named *rainbow table*. To avoid these kinds of attacks we must insert an unexpected variation, named *salt*.

Storing password on **client-side:**

- Should be only in user's head;
- If passwords are a lot, use an encrypted file or a password wallet.

The "Dictionary attack"

If you store plain hash of password, *dictionary attacks* are possible. This is possible under two hypotheses:

1. **known hash algorithm** that the server used;
2. leakage of information, so that the attacker has got a copy of the **password hash values**.

Hashes are not invertible function, but it is possible to make a **pre-computation** so maybe if there is no yet copy of any hash of password it is possible to decide that it would be nice in the future to attack passwords stored as plain SHA1 hash.

You must get a dictionary containing not only Italian languages but all the possible languages. For each word in the dictionary, you compute the hash of the word and store it in a DB paired with the corresponding word; with "word" we mean a possible passphrase, not a part of it. Typically, the attackers have dictionary extended to words such as names of famous people.

The main hypothesis is that the user chase one of the words contained in the dictionary. The attack is the following:

1. At some point the attacker gets a hash value, due to a leakage;
2. The attacker performs a simple **lookup** as following: $w = lookup(DB, HP)$, where DB is the database and HP is the computed hash of a word in the database, if any of the hash password appears in any tuple;
3. If the response is positive, the password is equal to that word. If not, the password is not the dictionary used.

Pre-computation is the key, because if you wait until you get a copy of the hash of password, and only at that point you start computing all the possible hash value, it could be too late, because the password could have changed.

Rainbow table

Dictionary attack can be made faster and more effective by the **Rainbow table** technique. It is still a *dictionary attack*, but it is a trade-off between space and time. Trying all possible password, computing the hash would be fast, but then the result will be a huge database. If you have a complete database the lookup would be fast, but here less passwords are stored and a bit more time is taken to compute the password if the corresponding hash is present. This is an improvement because makes exhaustive attack feasible for certain password sets.

Imagine creating a rainbow table to attack a password that we know contains 12 digits. The exhaustive attack would require 10^{12} rows that is a huge number of lines (containing passwords and the corresponding hash values). Rainbow table could be used to reduce the number of rows in the database by a factor 1000. In this way, we get a 10^9 rows database, where each line represents 1000 passwords. To achieve that we use the **reduction function:**

$$\boldsymbol{r}: \boldsymbol{h} \rightarrow \boldsymbol{p}$$

It is a function r that takes as input a hash and creates one possible password. **Beware that this is NOT the inverse of the hash,** because the inverse of hash does not exist. It is just a *mapping function* that from a hash creates one of the possible passwords of the whole set.

Then, the **pre-computation** is the following:

1. Select 10^9 distinct passwords (the wanted size) called *P;*
2. For each of them initialize the computation starting from that specific password, and then iterate for 1000 times: each time the hash of the current password is computed (called *k*) and then the *reduction function* is used to go from *k* to another possible password;
3. At the end, the password *P* of the first cycle is stored in the database together with the last computation of the reduction function (called *p*).
4. Then, the entry implicitly represents all the 1000 password tried. Note that there is no more hash to be stored.

- **pre-computation:**
  - for (10^9 distinct P)
    - for (p=P, n=0; n<1000; n++)
      - k = h(p); p = r(k);
    - store (DB, P, p) // chain head and tail

Then the **attack** rises in this way:

1. HP is the leaked hash of a password;
2. Start an iteration of 1000 times at most, and every time the *reduction function* is used to go from the value of the hash to a possible password;
3. Next, search the database to see if there is a row where p (the function calculation) is the end of the chain. In that case, we found the chain containing that hash, otherwise a new value k is calculated by performing the password hash;
4. After finding the chain, the computation of the hashes must be done again to see which is the hash that match the one that we have.

- **attack:**
  - let HP be the hash of a password
  - for (k=HP; n=0; n<1000; n++)
    - p = r(k)
    - if lookup(DB, x, p) then exit ("chain found!")
    - k = h(p)
  - exit ("HP is not in any chain of mine")

The problem is that since the *reduction function* is going from a hash to one possible password, there could be two different hashes that generates the same password, and this is called **fusion**. Rather than using a reduction function, a set of *n* reduction functions is used, one for each reduction step. On internet there are on sale pre-computed rainbow tables for various hash functions and password sets (e.g., SHA1 for alphanumeric).

This technique is used by various attack programs.

Using salt in storing password

The critical point in previous kind of attack is that the attacker performs pre-computation. Without the rainbow table and without the database created by the dictionary, it would take a lot of time. For this reason, we must avoid the pre-computation of the attacker, and that can be done with a simple trick: **do not give to the attacker the information to pre-computation**, because it is based on the idea that the attacker may know which is the password (through the dictionary).

Using the following technique, even if it can be possible to guess what a possible password is, the attacker does not get the hash table because every time a User ID is created, the system generates a **salt** that is different for each user. The salt is a random (unpredictable) and long (increased dictionary complexity) string of bytes. Users do not have to memorize the salt, which should contain rarely used or control characters. Then the hash is computed using the password concatenated with salt: $HP = hash(pwd \mid salt)$. The verifier stores *UID*, HP_{UID} and $salt_{UID}$. If someone gets the information in the database, he also gets the salt, but only then the computation can start, which will require a lot of time and in the meanwhile the password could have been changed. Additionally, there are different HP for users having the same password.

This makes the dictionary attacks nearly impossible (included those based on rainbow tables).

The LinkedIn attack

In June 2012 someone was able to copy 6.5 M password from Linked which were **unsalted, plain SHA-1 hash**. That person published those hashes on Internet and asked for *crowdsourcing* used for cooperative password cracking (which means: try to compute SHA-1 hashing of words and look if someone has a match). At least 236.578 password were found before the Interpol was able to ban the website that published the password hashes.

Simultaneously LinkedIn found out that the LinkedIn app for iPad/iPhone was sending in clear sensible data (not relevant to LinkedIn!).

Case History: passwords in MySQL

MySQL is a database where username and password are stored in the "user" table. MySQL (from v4.1) uses a **double hash** (**but no salt**!) to store password: $SHA1(SHA1(password))$. Then, the hex encoding of the result is stored, preceded by * (to distinguish this case from MySQL < 4.1). For example:

- For the password "Superman!!!" the field user.password is:

$$*\,868E8E4F0E782EA610A67B01E63EF04817F60005$$

To verify that this is the double hash of the word, you can use the following command on Linux:

```
$ echo -n 'Superman!!!'| openssl sha1 -binary | openssl sha1 -hex
(stdin)= 868e8e4f0e782ea610a67b01e63ef04817f60005
```

This is the standard way for MySQL to store password, which is no good. A good thing is to change the standard way MySQL uses to store password by using a salted one.

Case History: iPhone ransomware

In May 2014 iCloud account (with 1-factor authN) has been violated. Someone was able to discover passwords and then they once entered in the account used the *"Remote lock"* feature to **lock smartphones.** Then a message was sent to the victim which was saying: "*Device hacked by Oleg Pliss! To regain control, send 100 USD/EUR via PayPal to lock404@hotmail.com*".

The only possible way was to use the *Recovery Mode* which erases all data and applications. This was a destructive attack, since the attacker did not want to earn money (PayPal payments would have been traced however), but he asked people to pay even if the account was fake.

Strong (peer) authN

Recently emerged the obligation not to use standard authentication but to use strong peer authentication. It is always requested in specifications, but it is never formally defined, or it is defined in too many ways (which is useless).

ECB definition for Internet Banking

According to ECB (European Central Bank) a strong customer authN is a procedure based on the use of two or more of *knowledge*, *ownership*, and *inherence*. The elements selected must be mutually independent, so that the breach of one does not compromise the other(s). At least one element should be non-reusable and non-replicable (except for inherence), and not capable of being surreptitiously stolen via the Internet. The strong authentication procedure should be designed in such a way as to protect the confidentiality of the authentication data (if you use a password, you cannot send it in clear).

PCI-DSS definition for payment with credit cards

According to PCI-DSS definition, which is for payment with credit cards, starting from v3.2 it requires **multi-factor authentication (MFA)** for access into the cardholder data environment (CDE): it does not matter if it a trusted or untrusted network, and it is also used when the access is performed by administrators. The only exception is with *direct console access* (physical security), which means that you enter the room where the

server is placed. For remote access it is always required from untrusted network and by users and third-parties (such as maintenance).

This was the best practise until Jan '18 and it has been made compulsory afterwards.

***Remember**: MFA is **not** twice the same factor (e.g., two passwords).*

Other definitions

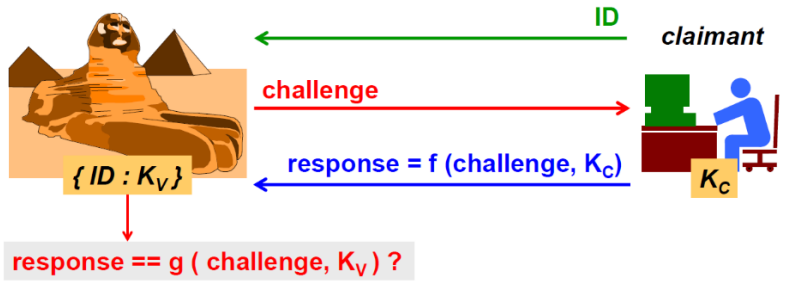
According to the *Handbook of Applied Cryptography*, authentication is a **cryptographic challenge-response identification protocol.** More in general, it is a technique resisting to a well-defined set of attacks. An authN technique can be regarded as strong or weak depending on the attack model:

- E.g., users of Internet Banking → ECB definition;
- E.g., employees of PSP → PCI-DSS definition.

In general, watch out for your specific application field because that is telling you the kind of risks and the kind of strength that your strong authentication should have.

Challenge-response authentication (CRA)

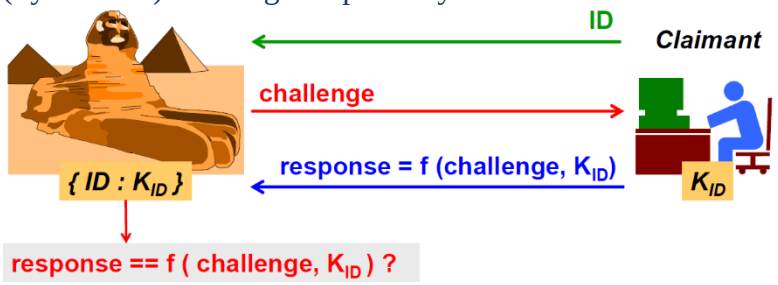
Challenge-response protocol is a possible way to implement strong authentication. CRA means that there is a challenge sent to the *Claimant* from the Verifier. The Claimant replies with the solution computed using some secret knowledge and the challenge. The Verifier compares the response with a solution computed via a secret associated to the Claimant.



Someone claims to own the identifier (*ID*). The verifier looks that there is a row associated to that ID and sends to the claimant the *challenge*. The claimant has a key (K_c) and uses it to perform some kind of computation (function f) and generates a response. The response can be checked by applying the function g to the challenge and to a well-known key (K_V) of the Verifier. The keys can be different or the same.

- The challenge must be **non-repeatable** to avoid replay attacks. For this reason, usually the challenge is a (random) *nonce*.
- The function f must be **non-invertible**, otherwise a listener can record the traffic and easily find the shared secret by using the function $K_c = f^{-1}(response, challenge)$

(Symmetric) challenge-response systems



In this case, there is a common key shared between Claimant and Verifier, which is typically the password or passphrase of user. The function f in computed two times: once from the user to make the response, and once from the verifier to verify the match.

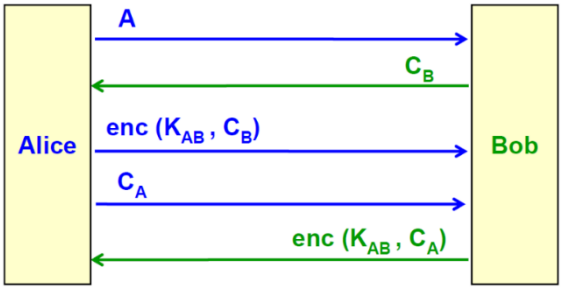
General issues with **Symmetric CRA** are:

- The easiest implementation uses a hash function (faster than encryption) such as SHA1 (deprecated), SHA2 (recommended) or SHA3 (future);
- K_{ID} must be known in cleartext to the Verifier and this may lead to attacks against the $\{ID : K_{ID}\}$ table at the Verifier;

There is a technique called ***SCRAM*** (**S**alted **C**RA **M**echanism) which solves this problem by using hashed passwords at the Verifier, which also offers channel binding and mutual authentication, while we are always talking about single authentication.

Mutual authentication with symmetric challenge (v1)

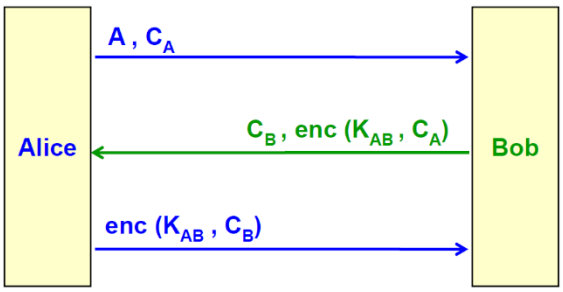
Imagine that we are not using hash but encryption (with hash would work in the same way) and imagine that we want mutual authentication.



Alice and Bob have a shared key K_{AB} and Alice sends to Bob a message *A* (which means: "Hey, I'm Alice!"). Bob replies with a challenge C_B and Alice response with the $enc(K_{AB}, C_B)$ which is the encryption of the challenge using the shared key. Then Alice could make a challenge too (C_A) and Bob would answer in the same way by doing $enc(K_{AB}, C_A)$. In this way there is protection against MITM attack, because if the challenges are nonces the Replay Attack is not possible.

Mutual authentication with symmetric challenge (v2)

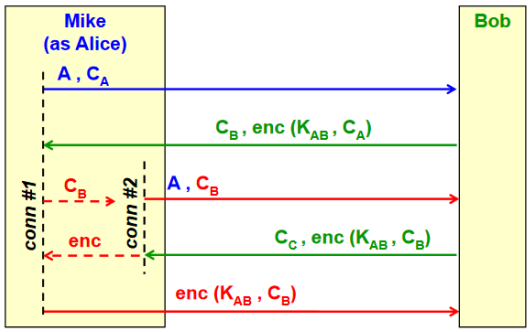
IBM in its SNA network system used the same technique with a different implementation: they reduced the number of messages to have better performance but no impact on security.



In the first step Alice send both the identity (*A*) and challenge (C_A). The response from Bob has both the challenge (C_B) and the encryption $enc(K_{AB}, C_A)$. At last, Alice replies with the $enc(K_{AB}, C_B)$. This seems equivalent to the previous solution in terms of functionality and security, but it is not.

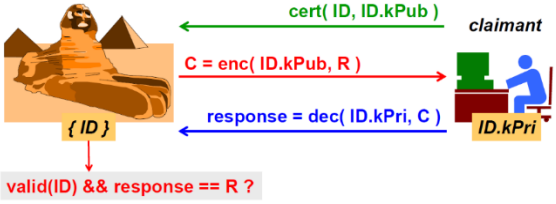
Attack to the symmetric challenge protocol (v2)

Here there is Mike pretending to be Alice. Mike sends to Bob the Alice's identity (*A*) and Alice's challenge (C_A). Bob replies with C_B and $enc(K_{AB}, C_A)$.



At this point, Mike does not know K_{AB} and cannot compute the response to the challenge. But at this point Mike opens a new connection with Bob, sending again Alice's identity (*A*) but this time sends the challenge sent from Bob (C_B). Bob replies again with another challenge C_C and $enc(K_{AB}, C_B)$ which is the answer to the challenge of the 1st connection. Mike can finally provide the correct answer. Of course, this can be countered if there is a limit in connections.

(Asymmetric) challenge-response systems



It is possible to also use an *asymmetric* challenge-response. This time the user does not send his identity, but its *X.509* certificate, which is declaring his identity and public key (the claimant has the corresponding private key *ID.kPri*). The Verifier creates a challenge by taking a random nonce *R* encrypted with public key. The claimant can decrypt it by using its private key and send it back to the Verifier.

If the response is equal to the *R* value put in the challenge, then the possession of the private key is proven. The only needed check is to verify if a valid ID is register in the Verifier's database.

Analysis

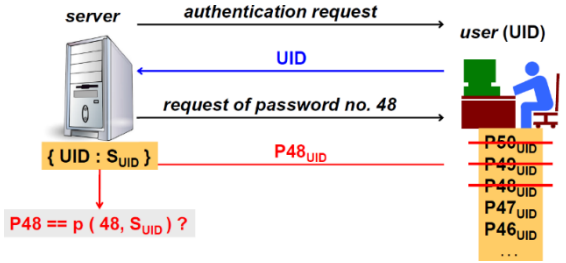
Asymmetric CRA is the strongest mechanism usable, which does not require the storage of any secret at the Verifier. This is always implemented for *peer authentication* (client and server) in *IPsec*, *SSH* and *TLS*. This is also the cornerstone for user authentication of a new authentication system called *FIDO*.

It has some problems such as:

- **Slowness**, that derives from the fact that asymmetric cryptography is generally slow;
- If designed inaccurately may lead to an **involuntary signature by the Claimant** because the claimant is getting something and it is performing a computation with private key, but the computation with private key has two applications: one for decrypting, one to create a signature. So, there is the risk to sign a document. For this reason, the data given as input to decryption claimant-side must have a specific format;
- **PKI issues** (*trusted root*, *name constraint*, *revocation*) because it uses certificates. This is avoidable if the Verifier stores kPubID (moves equivalent PKI effort to the Verifier). This is the approach followed by SSH: take public key and public it on the server, with a big risk if someone can change the key stored on the server.

One-time password (OTP)

Rather than having reusable passwords, users have passwords valid only for one login. The claimant that wants to access the server receives an *authentication request*. The user replies by using it is userID. At this point the server request a specific password among a long list (e.g., no. 48). If $P49_{UID}$ and $P50_{UID}$ have already been used, the user sends $P48_{UID}$ and set it as used on the table.



At this point the Server has a table containing the users (*UID*) and the secret (S_{UID}) that the user does not know. The secret was used to generate passwords. At this point, the Verifier will perform the same function used to generate password to generate it again and check if it matches with the one sent from user. This makes password **immune to sniffing**. Password can be sent in clear because next time it would be another one.

Summary

- Password is **valid only for one run** of the authentication protocol, next run requires another password;
- Hence, it is **immune to sniffing**;
- It is **subject to MITM** as someone that place the role of the verifier: in order to avoid this attack, Verifier authentication is needed;
- It has **difficult provisioning** to the subscribers: lot of passwords are required, and the password set will be empty at some point;
- There is also **difficult in password insertion** because it usually is a sequence of random character, to avoid guessing.

OTP provisioning to the users

If we assume that the claimant is working on a *stupid device*, that is it has not computation capabilities, or insecure/untrusted workstation, then in those cases some physical additional safety should be used. For example:

- Sheet of paper containing pre-computed passwords;
- Hardware authenticator (crypto token): an object where passwords are stored or generated.

On the contrary, if working on intelligent and secure/trusted workstation they can be **automatically computed** by an ad-hoc application (typical for smartphone, tablet, laptop, …).

The S/KEY system

This was the first OTP definition and implementation by Bell Labs (1981).

- The user generates a secret S_{ID} ;
- Then the user autonomously computes N one-time passwords where $P_1 = h(S_{ID}), P_2 = h(P_1), \dots, P_N = h(P_{N-1})$ (associated hash)
- The Verifier stores the last one P_N. This password will never be used directly for authentication, but only indirectly.
- When user wants to access the server, the Verifier (which has got P_N) asks for P_{N-1} and gets X from user. Then the following check is performed:

  $if\ (P_N != h(X))\ then$

  $\quad FAIL$

  $else$

  $\quad OK;\ store\ X\ as\ P_{N-1}$

The trick here is that the Verifier asks for passwords in **reverse order**. In this way the Verifier has no need to know the user's secret and only user must know all passwords. It is documented in *RFC-1760* and used MD4 (other choices are possible) and S/KEY is an example of Off-line / Pre-computed OTP.

***Note**: MITM is always possible for OTP, therefore S/KEY should be used in conjunction with **server authentication**.*

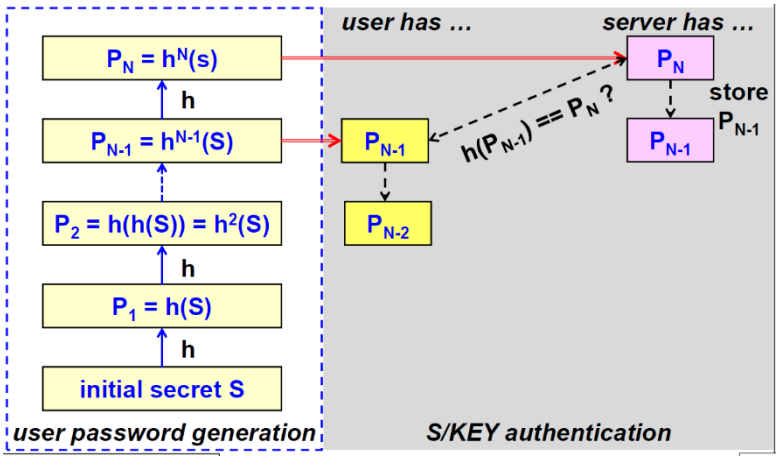


Figure 39 Password generation and verification using the S/KEY system

S/KEY – Generation of the password list

The user inserts a pass phrase (PP) which must be minimum 8 characters long and it must be secret, otherwise if it is disclosed then the security of S/KEY is compromised. The PP is concatenated with a server-provided seed that is not secret (sent in cleartext from S to C). This allows to use the same PP for multiple servers (using different seeds) because it is the combination between PP and seed, and it is also possible to safely reuse the same PP by changing the seed.

A 64-bit quantity is extracted from the MD4 hash that generates 128-bit (by XORing the first/third 32-bit groups and the second/fourth groups).

S/KEY – passwords

64-bit passwords are a compromise, but you have yet to enter them. Entering 64-bit as hexadecimal characters means 16 hexadecimal characters. So, normally they are entered as a sequence of six short English words chosen from a dictionary of 2048 ($e.g. 0 = \text{A}, 1 = \text{ABE}, 2 = ACE, 3 = ACT, 4 = AD, 5 = ADA$). It means selecting 11 bits from the computed hash and using a dictionary which have some simple words corresponding to the combinations (2048 in this case), and client and server **must share the same dictionary**. For example:

- **Password (text):** YOU SING A NICE OLD SONG, but not because it is a password but because "YOU" it is one of the words of the dictionary and it represent 11 bits. In total there are 6 words, which means 66-bits (more than 64, so it is good).
- **Password (numeric):** $1D6E5001884BD711$ (hex) or $2,120,720,442,049,943,313\ (decimal)$

This is just an example on how to encode in a user-friendly way a long bit string.

OTP Problems

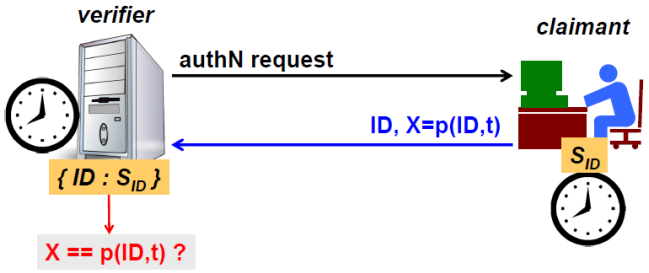
- OTP is generally **uncomfortable**: not easy to enter those strange passwords;
- **Uncomfortable when** used to **access multiple** password-based services (e.g., POP with periodic check of the mailbox);
- **Expensive when** based **on hardware authenticators:** nowadays it is also possible to use a software installed on a smartphone, but this raises the question of whether the smartphone is secure or not;
- **Paper-based** passwords cannot be used by a process but **only by a human operator;**
- Generation of good random passwords and beware of the hash function (MD4 is no longer good);
- **Password provisioning** (generator? SMS?);
- When **time** is used (as a base) to generate the OTP, there must be **time synchronization** between user and system: if they do not have the same time, there could be DoS (never able to login because always sending wrong password) or MITM, if the attacker makes the user believe that is 5pm, so the user uses that time to make the password and then the attacker uses it at 5pm.

Problems of hardware authenticators

Device creating passwords could have some problem:

- **Denial-of-service**: deliberately wrong attempts to trigger account blocking;
- **Social engineering**: if the authenticator is not a physical dedicated device, but it is instead a software on the smartphone of the user, it could be possible to make a phone call to simulate loss of the authenticator and remotely initialize a new one.

Time-based OTP



In this solution the password depends upon time and the user's secret: $p(ID,t) = h(t, S_{ID})$. When the claimant wants to authenticate receives a request from the verifier and it will submit its own ID plus the value generated by a device (authenticator) which tells the claimant which the correct password is to be entered now. At that point, the verifier needs to check if the value is the current OTP for the current time. Since verifier has a big table containing for each user the corresponding secret, it can perform the same computation and compare it.

This kind of OTP requires local computation at the subscriber and clock synchronization (or keeping track of time-shift for each subscriber). Due to the fact that the password needs time to be sent, the time needs to be quantized (that is, considering timeslots, usually long from 30 to 60 seconds) and an authentication window: usually this is done by the verifier considering to be correct the password as generated one time slot before and one after the correct time slot according to its own time: in formulas, the authentication succeeds if $X == p(ID,t) \,||\, X == p(ID,t-1) \,||\, X == p(ID,t+1)$ (where t is the timeslot). Typically, only one authentication run per timeslot, that could be not good for some services, for example a broker that needs to perform many transactions at a time.

This system is prone to time attacks against subscriber and Verifier: usually servers and clients get the time from an external source and an attack can be performed using fake NTP server or mobile network femtocell. For example, if the attacker manages to trick the subscriber to give to him a password for a future timeslot, he can then use it at that time.

Since the Verifier stores the secret for each user, it must handle a very sensitive database: if it gets stolen, then the attacker can impersonate and compute the password of any user. An attack of this type happened against a TOTP system called *RSA SecurID*.

A TOTP example: RSA SecurID

In this system the Claimant sends to the Verifier in clear the triple $\{user, PIN, tokencode(seed, time)\}$: since RSA SecurID is a physical device which continuously displays the correct password, if someone different from the subscriber reads it and then uses it, then an attacker could enters with the subscriber's identity, which means that another factor for authentication is needed. For this reason, one factor is "*owning the device*" while the other is "*knowing the (reusable) password*".

Since it could be possible for the PIN to be sniffed while sent across the network, there is a variant of the SecurID authenticator that includes a "pin pad" (a sort of small keyboard just for entering the PIN), and when the PIN is entered, the value of the PIN is taken into the account for the generation of the OTP. In this case, it is possible to send just the user and the modified token code (the tuple {*user, token-code*(seed, time, PIN)}*), which is also function of the PIN. Based on user and PIN, the Verifier checks against three possible token-codes: TC_{-1}, TC_0, TC_{+1}, as any TOTP system.

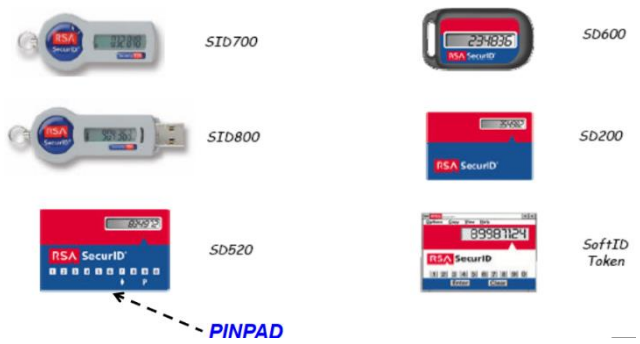


Figure 40 Different authenticators provided by RSA: the SID700 is the basic model; the SID800is provided with an USB interface to let programs automatically read the code; the SD520 is a credit card size device provided with a pin pad; the Soft ID Token is a software application that performs the same computation of the hardware devices, to be used in case the used device can be trusted.

Moreover, each device is associated with two PINs: the first one is the one used normally for authentication, while the second is called ***duress code***, and it is used to generate an alarm, when the subscriber is under attack (for example when the user is forced by a criminal to authenticate).

SecurID: architecture

Besides providing the hardware authenticator which computes the token code, RSA also provides some component called **ACE** (Access Control Engine):

- **ACE client** is installed at the Relying Party, which is the server that wants to use the authentication system;
- **ACE server** implements the Verifier.

In the picture, the ACE server at the top is the verifier an so it has the global information to validate a token code. The servers for the service are the Relaying Party, the parties that have access control to be implemented with RSA SecurID: they must have an implementation of the ACE client, that is needed to communicate with the verifier, to validate the credential submitted by the user. In the example at the left side, there is a remote access to a server via SSH: the user having a normal SecurID device provides the triple *{user, PIN, TC}*, then when this information is received by the Relaying Party, it is forwarded to the verifier, which in turn will provide an answer (credentials valid or not). At the right side, there is an example of failed authentication using the device with pin pad.

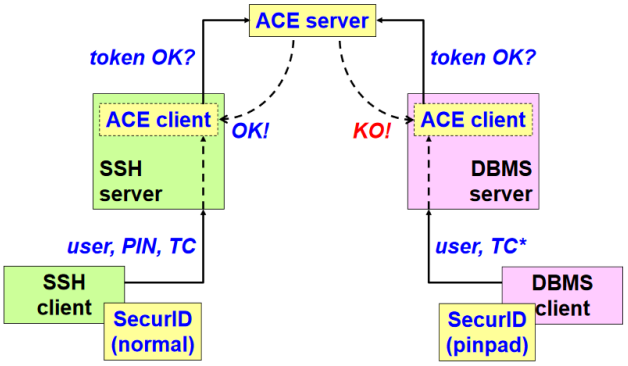


Figure 41 Structure of the SecurID architecture

It is important to look at this architecture not only as the implementation of SecurID, but also as a general architecture from which is possible to see that if some special way of authentication is needed, most often it is necessary to enhance the servers with some software that is able to handle that new type of authentication (in the example the DBMS software). If a company wants to share the same authentication technology across

multiple servers, most likely a centralized verifier is needed, which provides verification for all relaying parties: in this sense it is possible to compare this schema with the NIST one.

Event-based OTP

The principal problem of TOTP is that only one authentication per timeslot is allowed. This solution uses a monotonic integer counter C as input besides the seed: $p(ID, C) = h(C, S_{ID})$. It also requires local computation at the subscriber, and the counter is incremented at the subscriber (e.g., through a button): in this way frequent authentication runs are possible. This system permits OTP pre-computation (also by adversary who temporarily have access to the authenticator): at the same time, Verifier must accommodate desynchronization, because for example the subscriber pushed the button unwillingly. To do this, a counter window is considered, so that a password is considered correct if it corresponds to the one computed with one of a set of counters, with a fixed and usually small set size. In formulas, a password is considered correct if $X == p(ID, C) \,||\, X == p(ID, C+1) \,||\, X == p(ID, C+2) \,||$ ... with at most ten subsequent counters, to be resistant to exhaustive attacks. The Verifiers can accommodate desynchronization and each time store what was the counter that matched with the password sent by the user. If for any reason none of the passwords generated user side is ok, then a call to reset the counter will be needed (for example imagine a child taking the device and pressing the button more than ten times).

Out-of-band OTP

The solutions illustrated so far need the user to receive something: a list of passwords in the case of S/KEY or a device in case of TOTP and EOTP: in case these options are not good, for example because it is not possible to provide securely the list or there is no trust on the device of the user, it is still possible to use Out-of-band OTP. It is based on the fact that the OTP is generated not using the normal communication channel. In the schema in the figure, the user is provided with a secret key, that is a *reusable password*. While authenticating, the user sends the user id and the reusable password: this is needed to clearly identify who really the user is. The reusable passwords, as seen, are not very strong but thanks to the third step the verifier can be sure that the user is really authenticating: it looks up into its database and retrieve the registered phone number; then it generates an OTP and send it (out of band) to the user's cell phone, via SMS for example (step four). This, as it is possible to see, is done out of band because the OTP transmission uses a different medium than the communication between the server and the user. Finally, in step five, the user can provide the OTP just received.

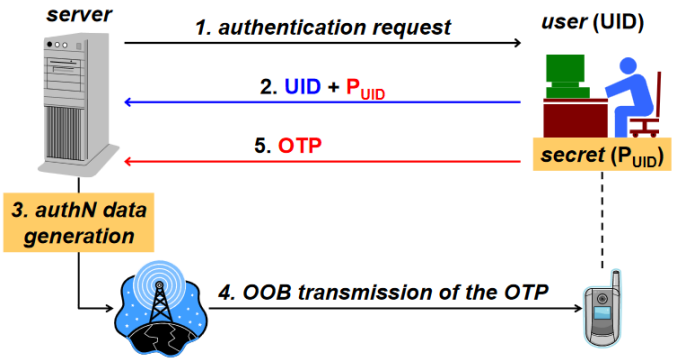


Figure 42 The steps of an Out-of-band authentication schema

This is a widely used system by many banks and many providers of identity like the ones involved in SPID. This system also reduces the burden on the user, who does not need to have a modern smartphone, but just a minimal device to receive the message: the OTP is generated by the server only when needed and then sent in a trusted way to the user. Beware however that at step five secure channel with server authentication is needed to avoid MITM attacks. OOB channel frequently is text via SMS message but, since most mobile networks are nowadays implemented with VoIP, mobile user identification, and SS7 protocol (which are quite insecure), the NIST suggests using Push mechanism over TLS channel to registered subscriber device, by putting the message inside a notification, the confirmation of which is made by inserting the user's fingerprint.

Two-/Multi-Factors AuthN (2FA/MFA)

It has already pointed out that, if we want to provide strong authentication, more than one factor should be used: this is also named **2FA** or more in general **MFA**. MFA is used both to *increase authN strength* and to *protect the physical authenticator*, as seen for OTP. Usually, a **PIN** is used for authenticator protection:

- **PIN transmitted along with OTP**: as seen, the possible problem is that it can be sniffed when entered or transmitted across the network;
- **PIN entered to compute the OTP itself**: like in *RSA SecurID*;
- **PIN** (or inherence factor) **used to unlock the authenticator**, very risky if:
  - no protection from **multiple unlock attempts**;
  - **lock mechanism weak**, for example the fingerprint sensor is not enough strong to recognize only the device owner's fingerprint;
  - **unlocking valid for a time window**: if someone has access to the authenticator in the time window in which it is unlocked, then he can use the user identity.

Authentication of human beings

Another problem that one could want to address is to verify that the subscriber is a human being rather than a program. There are two solutions:

- **CAPTCHA** techniques (**C**ompletely **A**utomated **P**ublic **T**uring test to tell **C**omputers and **H**umans **A**part): for example, a picture with images of distorted characters;
- biometric techniques: for example, fingerprint.

Biometric systems and their problems

The main idea is to measure one biologic characteristics of the user, such as: fingerprint, voice, retinal scan, iris scan, hands' blood vein pattern, heart rate and hand geometry. However, whatever is the adopted solution, there are two parameters that cannot never be equal to zero:

- **FAR** (**F**alse **A**cceptance **R**ate): the rate at which the system accepts as good a biometric signal that actually it's not;
- **FRR** (**F**alse **R**ejection **R**ate): the rate at which the system rejects a valid signal as if was fake.

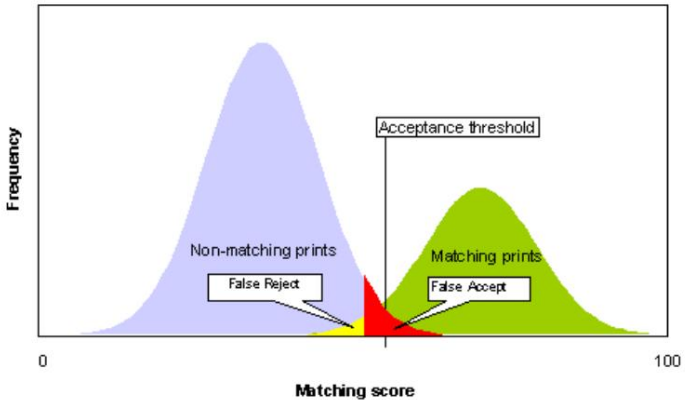


Figure 43 Problems of biometric systems: the two normal distributions are referred to fingerprints that match (green) and those that do not. The two distributions overlap, and this means that it is needed to put a threshold: wherever it is put, FAR and FRR cannot both be equal to zero.

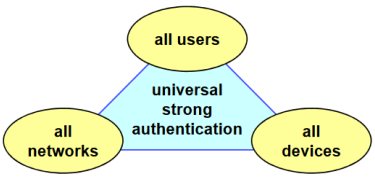
Those rates can never be equal to zero, because biometric systems are inherently imprecise: FAR and FRR may be partly tuned but they heavily depend on the cost of the device. Moreover, some biological characteristics are variable: for example, the user can have a finger wound, the voice altered due to emotion or retinal blood pattern altered due to alcohol or drug. Other than technical ones, there are also other kinds of problems:

- **psychological acceptance**:
  - **"Big Brother" syndrome**, related to personal data collection;
  - some technologies are intrusive and could harm, for example retinal scans can be not good for some people.
- **privacy**, because at this point it is not a matter of authentication, but of identification and it cannot be repudiated;
- **cannot be changed if copied**: a password can be changed if there is the fear that it has been disclosed, while a biometrical feature cannot. For this reason, a biometric authentication should not be sent across the network, it can only be useful to locally replace a PIN or a password;
- lack of a standard API / SPI that leads to:
  - high development costs;
  - heavy dependence on single/few vendors.

Authentication interoperability: OATH

As said, companies managing a lot of servers would like to use the same authentication for all their relying parties (that is, the application servers), but then it means that should be possible to buy a verifier and use it with devices created by different companies. In the past there was not such interoperability, so for example when buying RSA SecurID devices, it was needed to also buy the ACE client and the ACE server from RSA. RSA did not even publish the hash algorithm used in the generation of token-code.

To solve this problem, there was an initiative called OATH: the idea is to provide interoperability of authentication systems based on OTP and symmetric or asymmetric challenge. This is done by the development of standards for the client-server protocol and the data format on the client, in order to achieve universal strong authentication.



So far, OATH provided the following specifications:

- **HOTP** (HMAC OTP, RFC-4226);
- **TOTP** (Time-based OTP, RFC-6238);
- **OATH challenge-response protocol** (OCRA, RFC-6287);
- **Portable Symmetric Key Container** (PSKC, RFC-6030): while using OTP, the shared secret must be protected. This solution provides an XML-based key container for transporting symmetric keys and key-related metadata;
- **Dynamic Symmetric Key Provisioning Protocol** (DSKPP, RFC-6063): it is a client-server protocol for provisioning symmetric keys to a crypto-engine by a key-provisioning server.

HOTP

Let us define:

- **K**: shared secret key (between Verifier and Subscriber);
- **C**: counter (monotonic positive integer number);
- **h**: cryptographic hash function (by default is *SHA1*);
- **sel**: function to select 4 bytes out of a long byte string.

The **HMAC-based OTP** is computed in the following way:

$$HOTP(K,C) = sel(HMAC - h(K,C)) \,\&\, 0x7\text{FFFFFFF}$$

where the mask $0x7FFFFFFF$ is used to set MSB=0 (to avoid problems if the result is interpreted as a signed integer). Then to generate a N digits (6-8) access code:

$$HOTP - code = HOTP(K,C) \; mod \; 10^N$$

In this way we implement an event-based OTP.

TOTP

In order to create a Time-based OTP, the way is the same as HOTP but the counter C is the number of intervals TS elapsed since a fixed origin T0, in formulas:

$$C = (T - T0) \,/\, TS$$

Where, in the default (RFC-6238), ***T0*** is the Unix epoch (*1/1/1970*), ***T*** is $unixtime(now)$ seconds elapsed since the Unix epoch, ***TS*** is equal to *30 seconds*, ***h*** is SHA1 (buy may use SHA-256 or SHA-512) and ***N*** is equal to 6.

The important point is that with the same base functions it is possible to implement TOTP and EOTP.

Moreover, this standard is important because Google provides free open-source implementations of HOTP and TOTP, both for the client and the server. In Google's implementation, **K** is provided *base-32 encoded* (most

often as a QR code), **C** is provided as *uint_64*, **TS** is equal to *30 seconds*, **N** is equal to *6* and the function $sel(X)$ is such that: an offset with the 4 least-significant-bits of X is considered, then it returns $X[offset \dots offset + 3]$. If the generated code contains less than 6 digits, then it is left padded with zeroes (e.g., 123 → 000123).

The availability of this code from Google has pushed the usage of this standard, nowadays widely used.

FIDO

One of the most recent attempts to increase the security of authentication is the standard named FIDO (**F**ast **ID**entity **O**nline). It is an industry standard of the FIDO Alliance for:

- biometric authN, that in FIDO terminology is called *passwordless user experience;*
- 2-factor authN, that in FIDO terminology is called *2nd factor user experience.*

So, the focus of FIDO is to make the user experience in authentication simple, permitting them to use biometric system and more than one factor. The main concept of FIDO is not to invent another device, but let the user exploit any personal device which is capable of asymmetric cryptography (e.g., laptop or smartphone), because inside FIDO it will be used:

- for responding to an asymmetric challenge;
- for digital signature of texts.

The FIDO protocols are designed from the ground up to protect user privacy. The protocols do not provide information that can be used by different online services to collaborate and track a user across the services. Biometric information, if used, never leaves the user's device.

FIDO Registration

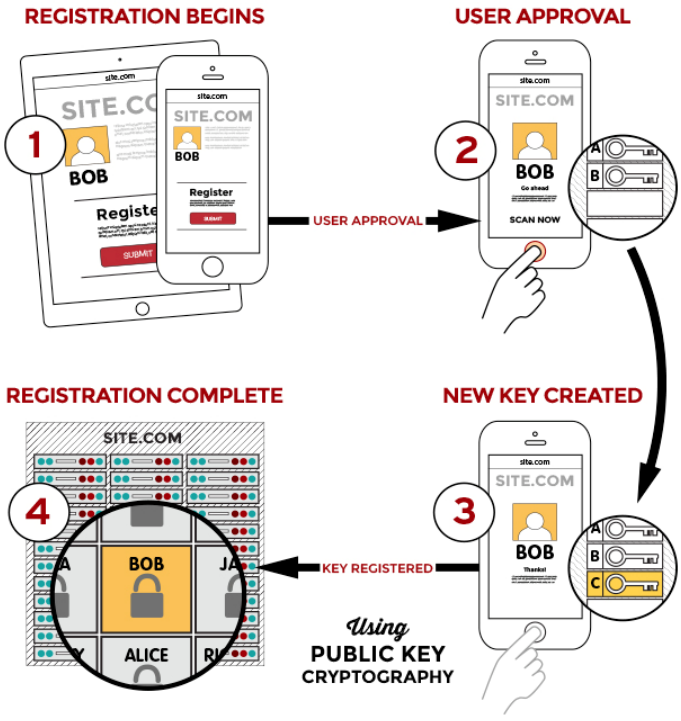


Figure 44 FIDO registration procedure, image taken from https://fidoalliance.org/how-fido-works/

Differently from other systems, in which registration is usually performed by a new classical username and password creation, or by reusing the user's Facebook or Google identity, FIDO uses the user's device with asymmetric cryptography. When registering with FIDO:

1. User is prompted to choose an available FIDO authenticator that matches the online service's acceptance policy.
2. User unlocks the FIDO authenticator using a fingerprint reader, a button on a second–factor device, securely–entered PIN or other method.
3. User's device creates a new public/private key pair unique for the local device, online service, and user's account.
4. Public key is sent to the online service and associated with the user's account. The private key and any information about the local authentication method (such as biometric measurements or templates) never leave the local device.

Please note that the real user's identity can be fictitious! However, the real identity of the user is not important, only the key is: the private key is stored inside the device and the public key in sent to the web server and it is associated to the name decided for authentication. There is the use of asymmetric cryptography but no X509 certificates (it is not needed), because the public key is stored at the server, associated with a name.

FIDO Login

After signing up, there are four steps to be performed to log in:

1. Online service challenges the user to login with a previously registered device that matches the service's acceptance policy: in this phase username and (reusable) password are provided by the user;
2. User unlocks the FIDO authenticator using the same method as at Registration time.
3. Device uses the user's account identifier provided by the service to select the correct key and sign the service's challenge: this identifier is retrieved by considering the pair username-password;
4. Client device sends the signed challenge back to the service, which verifies it with the stored public key and logs in the user.

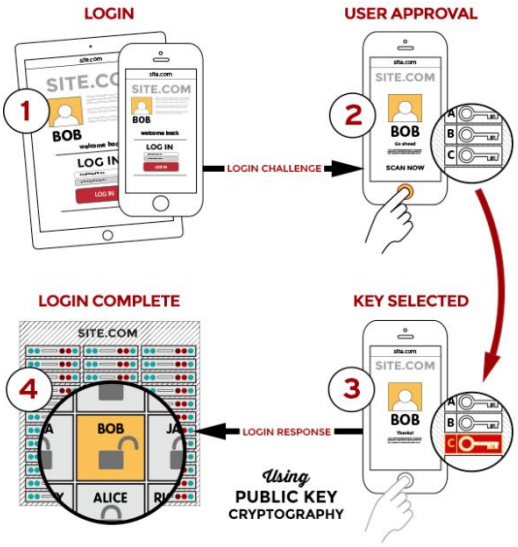
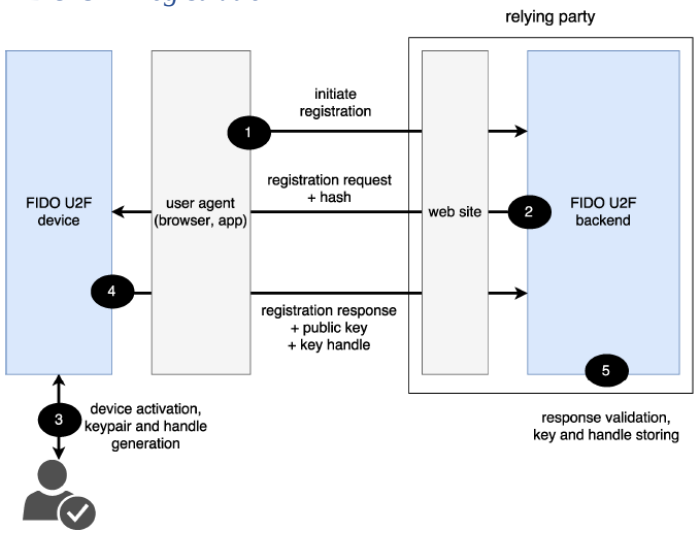


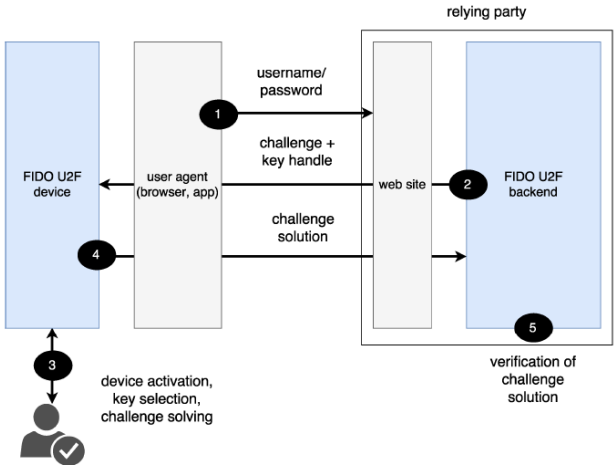
Figure 45 FIDO login procedure, image taken from https://fidoalliance.org/how-fido-works/

FIDO U2F registration



In the registration phase, typically the work starts with the browser or with an application that typically communicate with the server. The user registration phase means that the website on which the users try to sign up **must** have a FIDO U2F backend, which is something that can manage the FIDO protocol. When registration request arrives, the backend will send a registration request at the device including a hash (like nonce, to protect integrity of transmission). The browser/app will pass it to the FIDO U2F device (which can be software or hardware) and then the user must explicitly **activate the device**, authorize the **keypair generation** and the **handle generation**, which is an identifier for this specific key (typically the hash of the public key). Then the registration response is sent along with the public key and the key handle. Those items are validated in the final step (5) and the pair key-handle is stored.

FIDO U2F authentication



The authentication typically starts with the username and (reusable) password that are sent to the FIDO U2F backend, which will start a challenge that has to be solved with the key associated to the username. When the user receives it, the device must be again activated, the key must be selected (according to the key handle submitted by the server) and solve the challenge. Then the challenge will be solved and sent to the server for verification.

FIDO: other characteristics & security analysis

FIDO uses:

- **Biometric techniques**: local authentication method to enable the FIDO keys stored only on the user device;
- **Secure transactions**: digital signature of a transaction text (in addition to the response to the challenge) with the same key used for authentication. This is done to avoid MITM attacks;
- **FIDO backend (or server)**: to enable the use of FIDO on an application server
- **FIDO client**: to create and manage credentials FIDO on a user device

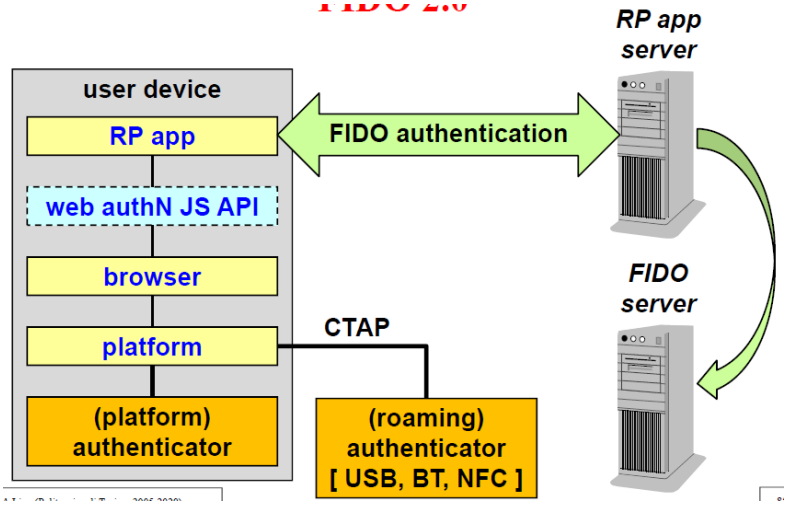
About security and privacy, FIDO provides:

- **Strong authentication** (*asymmetric cryptography*);
- **No 3rd party in the protocol**: since there is no use of X509 certificates. It's a sort of direct trust between user and web server;
- **No secrets on the server side**: not exposed to any confidentiality attack. Of course, since it stores public keys and identifiers of the users, authentication and integrity for those data is needed;
- **Biometric data** (if used) **never** leave user device;
- **No phishing because authN response cannot be reused**: it is a signature over the transaction text, including the RP identity (which means that the same response cannot be used on another server);
- Since one new keypair is generated at every registration, we obtain **no link-ability** among: different services used by the same user; different accounts owned by the same user.
- There is no limit in private key generated, because **private keys are not stored in the authenticator** but **recomputed** (generated on-the-fly) as needed based on an **internal secret** and **RP identity**. Using a proper internal function with the secret, it can be possible to use RP identity to compute private key.

FIDO evolution is the following:

- Feb 2013: FIDO alliance launched;
- Dec 2014: FIDO v1.0;
- Jun 2015: Bluetooth and NFC as transport for U2F;
- Nov 2015: submission to W3C of the Web API for accessing FIDO credentials;
- Feb 2016: W3C creates the Web Authentication WG to define a client-side API that provides strong authentication functionality to Web Applications, based on the FIDO Web API;
- Nov 2017: FIDO v2.0.

FIDO 2.0



With FIDO 2.0 there's the definition of a new protocol, the **CTAP (Client To Authenticator Protocol)** used to connect an external authenticator to the platform. When authentication starts, the RP application can use the *web authN JS API* to access FIDO. The API is running inside the browser, on top of a platform (such as Linux, iOS, Android) and then there are two choices: **local authenticator** (such as private key stored on a file on Windows) or an **external device** which is paired with the device with the **CTAP protocol**. This protocol runs through USB, BT, NFC, etc., and keys are always there. This is a kind of *roaming authenticator* because the device can be used with multiple application devices.

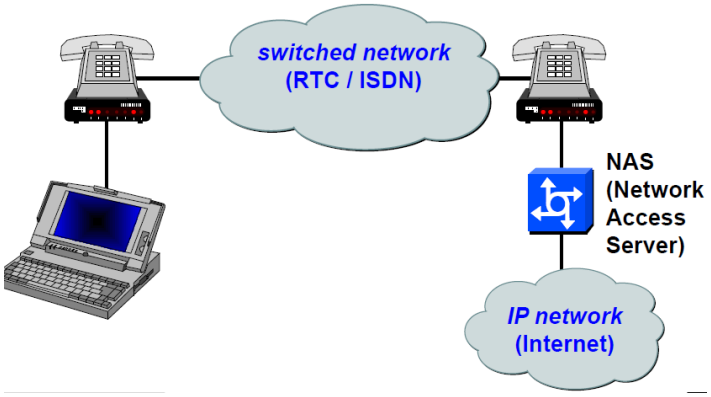
In case there is the use of internal authenticator, there are some cryptographic elements (more or less secure) able to store and use asymmetric keys.

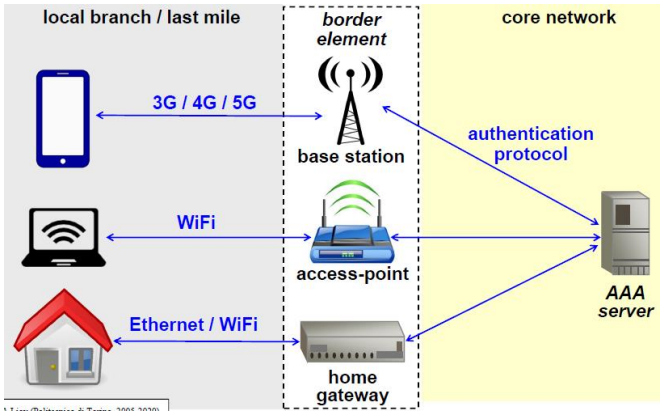
For example:

- **Packed attestation** is an authenticator with some limited resources (e.g., a Secure Element in Android);
- **TPM attestation:** a special chip which provides some good security features, can generate and use asymmetric keys;
- **Android Key attestation:** authenticator available from Android Nougat on;
- **Android SafetyNet attestation:** authenticator of Android via SafetyNet API;
- **FIDO U2F attestation:** authenticator FIDO U2F using the FIDO-U2F Message Format.

Security of IP networks

The first point about security of IP networks is about controlling who can access the networks. In the past, it was used mostly for residential users, by using a modem to employ a telephone line (switched network) to transform bits in sound and have an equivalent equipment on the ISP side, that would accept telephone call and transform it into network packets. To make this possible, there were some devices called **NAS** (*Network Access Server)* that had the task to authenticate user, perform access control and then provide to the user access to the IP network (typically internet, but in some cases also to access from home the internal network of a company). Nowadays this schema, at least in western countries, is not used anymore, even if some countries do still use it.





Today, it is possible to access Internet in different ways, that basically depends upon the device. For example, a smartphone typically uses some technologies such as 3G, 4G or 5G to connect to the base station, which runs an **authentication protocol** with the **AAA server** to check if user is authorized to internet connection. It is also possible to use Wi-Fi to the access-points, which provides the translation from wireless to wired network and again can verify the access. Otherwise, there could be a home gateway, which can provide the access to Internet by using both Ethernet and Wi-Fi (only if properly authenticated). In all these schemas, it is always needed to perform authentication before permitting traffic from a specific user. So, there is a difference between *local branch/last mile*, *border element*, *core network*.

Authentication of PPP channels

There is the need of authenticating a user before a network transmission is enabled. Since the authentication starts when someone is trying to connect, there is surely already available the layer 1 (physical) for physical transmissions. Since we are working at a logic level, the L2 is also available (data layer connectivity). On top of data layer connectivity, normally runs a protocol which is specific for transporting some quantity of data which is usually the **PPP** (Point-to-point) **protocol** invented to *encapsulate network packets* such as layer 3 (e.g., IP) *and carry them over a PPP link*. That can be a physical PPP like the ISDN line or the telephone network or it can be a virtual layer too, such as when starting from home gateway there's access to ADSL by using **PPPoE** (*PPP over Ethernet*) to transport packets between home gateway and provider. Moreover, PPP is used to carry packets inside virtualized layer three connections with specific and "horrible" protocol named *L2TP* (Layer 2 Tunnel Protocol) which is a layer two incapsulated inside UDP, which violates all normal behaviours of a network. Anyway, once PPP is enabled there are many virtual point-to-point going from the device to an access point. PPP activates in three steps:

- **LCP** (Link Control Protocol): establish the ability to transmit data;
- **Authentication** (optional; *PAP*, *CHAP* or *EAP*);
- **L3 encapsulation** (for example *IPCP*: IP Control Protocol);

Authentication of a network access

There are three chances to authenticate network access over PPP:

- **PAP** (***P**assword **A**uthentication **P**rotocol)*: this is the oldest one; in this case user sends username and password in clear over the PPP channel. If someone is sniffing the channel, it will acquire the password;
- **CHAP** (***C**hallenge **H**andshake **A**uthentication **P**rotocol)*: uses a symmetric challenge response based on user's password. In this case password cannot be copied but channel is not protected. Rather than inventing other protocols tied to a specific authentication mechanism, it has been made a generalization introducing EAP;
- **EAP** (***E**xtensible **A**uthentication **P**rotocol)*: it is a protocol which **does not** implement any method. The authentication method is external: for example, can be used challenge response, OTP or TLS.

Nowadays PAP and CHAP should never be used, while EAP is the most widely adopted for authenticating network access.

EAP

This is the **PPP Extensible Authentication Protocol** (RFC-3748). It is a **flexible L2 authentication framework**. It is L2 because before getting access to internet (which is L3) you must authenticate. It has already predefined authentication mechanism such as: *MD5-challenge* which is symmetric and similar to CHAP, generic *OTP*, and *generic token card*. Other mechanisms have been added: RFC-2716 "PPP EAP TLS authentication protocol" and RFC-3579 "RADIUS support for EAP".

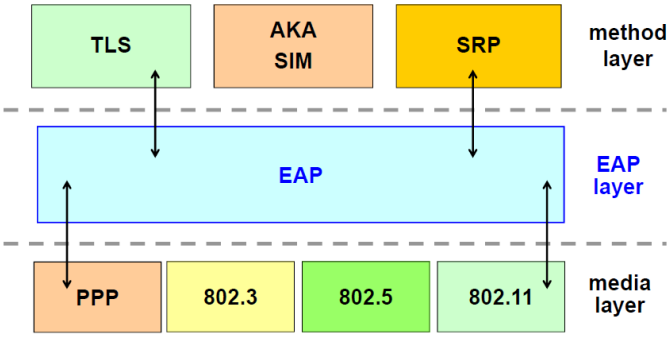
EAP needs to transport some data to perform the authentication (e.g., transport a username, challenge, or response) but L3 is not available (because EAP authenticates before L3). For this reason, EAP needs to create its own encapsulation protocol to transport its own data over L2.

- EAP defines some small L3 protocol used only by itself. The advantage is that this is completely **independent from IP** and it has been thought to support any link layer (EAP works over old PPP and the old 802.x).
- Since there is no L3, it provides **ACK/NAK** required for packets but there is no windowing (like it is provided in TCP). EAP assumes that packets are not reordered (in PPP it is guarantees from the protocol) but if you are using EAP over virtual channels like UDP and raw IP these datagrams may arrive out of order: in that case EAP would not work.
- **Retransmission** must be guaranteed, because packets need to be sent again if they are lost but there must be a limit in retransmission trials, typically between 3 to 5, after which the authentication fails.
- **No fragmentation** (which depends on the MTU underlying in the L2 network): must be taken care by the EAP methods for payload that are greater than the minimum EAP MTU.

When authentication does not work in EAP, it does not mean that authentication failed, but there could be a network problem. While troubleshooting EAP a network expert could be needed to check if any of these problems are the culprit of the failure.

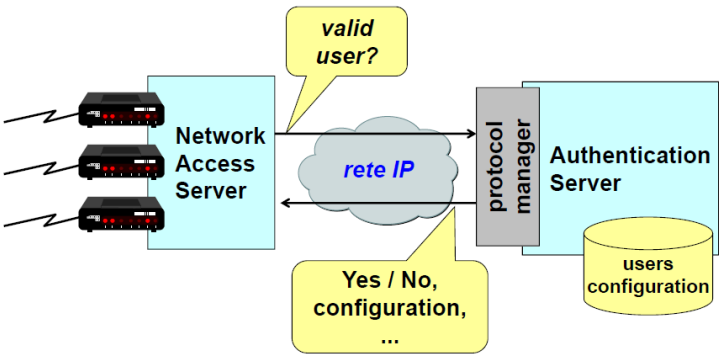
In EAP, the link is not assumed to be physically secure: any authentication method must provide security on their own. Some EAP methods are:

- **EAP-TLS** (RFC-5216)
- **EAP-MD5** (RFC-3748): symmetric challenge response which only provides EAP peer authentication, but no mutual authentication
- **EAP-TTLS**: tunneled TLS that permits to operate any method protected inside a secure TLS channel
- *EAP-SRP* (Secure Remote Password)
- *GSS_API* (includes Kerberos)
- *AKA-SIM* (RFC-4186, RFC-4187): **S**ubscriber **I**dentity **M**odule is the system used in mobile networks.



On the left, there is the general architecture of EAP. On the lower part there is any kind of L2 channel which is supported by EAP (e.g., PPP, 802.3 (Ethernet), 802.5 (Token ring), 802.11 (Wi-Fi)). EAP provides to those specific networks the availability of its authentication methods, such as TLS, AKA-SIM, SRP authentication (*independent* from the L2).

Authentication for network access



Authentication for network access works like the architecture on the right. On the left there are some communication links (modems, access points or ADSL/Fiber) which are at some point terminated in a device hosted by the ISP. Those devices (for connecting all possible users) are controlled by a NAS (*Network Access Server*) which receive requests from clients and needs to know if it is a valid user or not. It will use the protocol on the backend IP network local to the NAS and then will talk to the centralized authentication server. That is because ISP has got many points of presence with many NAS and they must all share the same information about users. The authentication server has access to a database that contains user credentials and the configuration for each user (depending on contract between user and ISP). The Authentication Server will response to the NAS with the response (valid/invalid user) and the configuration that the NAS must enforce on the traffic of the user.

NAS manufacturers claim that security needs three functions briefly named as **AAA**:

- ***Authentication***: entity's identity is authenticated based on credentials (e.g., password, OTP);
- ***Authorization***: determining whether an entity is authorized to perform a given activity or gain access to the resources or services;
- ***Accounting***: tracking network resource usage for audit support, capacity analysis or cost billing.

The AS performs exactly these three functions talking with one or more NAS via one or more protocols.

Network authentication protocols

Basically, there are three protocols which AS and NAS can use to communicate:

- **RADIUS**: it is the de-facto standard (and the most used one) with a nice feature that permits it to perform like a proxy towards other authentication systems. Can be both authentication system and use external AS;
- **DIAMETER**: it is the evolution of RADIUS and it puts the emphasis on roaming among different ISP and since it is more modern it has taken better care of security;
- **TACACS+ (TACACS, XTACACS)**: competitor of Radius, technically better but achieved lower acceptance because it was a proprietary solution, implemented only by Cisco with no public specification.

RADIUS

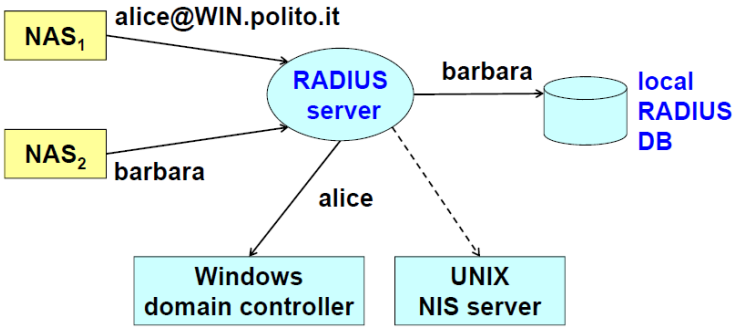
Radius stands for **Remote Authentication Dial-In User Service.** It is a very old protocol because it was considered for Dial-In (when users were dialing to connect to an ISP with modem), and it was invented nearly 30 years ago. It has evolved over time and **supports authentication, authorization, and accounting** to control network access:

- *Physical ports* (analogical, ISDN, IEEE 802): RADIUS can be used also to perform access control on Ethernet network.
- *Virtual ports* (tunnel, wireless access): to manage a network of access point in centralized way.

RADIUS as a concept implements **centralized administration and accounting** (to store all information about usage of resources). It is a **client-server** protocol between the NAS and AS and uses **port 1812/UDP** (authenticating) and **port 1813/UDP** (accounting). Since UDP is unreliable, each transmission of a RADIUS packet is subject to timeout. If no ACK is received after timeout, then the same packet is retransmitted and then there is a maximum number of appends after which the communication is declared impossible. RADIUS supports architectures with a main server but also a lot of **secondary servers**: this is useful to improve performance and resistance of the system (also to DoS attacks).

- RFC-2865 (Protocol)
- RFC-2866 (accounting)
- RFC-2867/2868 (tunnel accounting and attributes)
- RFC-2869 (extensions)
- RFC-3579 (RADIUS support for EAP)
- RFC-3580 (guidelines for 802.1X with RADIUS)

From the list above it is possible to see that from the basic protocol there have been a lot of extensions (also the EAP support) and in particular it supports also 802.1X which is a network access control security architecture.



The **RADIUS server** may act as a proxy towards other authentication servers. Looking at the picture, on the left there are two NAS, each one providing a request of access from users. RADIUS consults its *local RADIUS DB* on the right and will now check if (e.g., Barbara) is a valid registered user. In case it receives an access request maybe from another NAS in the form of an email address (although it is not an email address) *alice@WIN.polito.it* is an indication that the user Alice is not registered in the local RADIUS DB, but *Alice* is a user defined in the security domain *WIN.polito.it* to which the RADIUS server is associated somehow. This means that **RADIUS will act as a proxy for the authentication part** and will redirect the request to the *Windows domain controller*. Then the authorization/accounting could be managed locally by the RADIUS server. RADIUS can also be associated to another domain (e.g., *UNIX NIS server*).

Which security functionalities for Radius?

Radius needs security functionalities because of:

- **Sniffing NAS requests (if contains passwords):**
  - Problem of *confidentiality*, in the sense that if the request would contain the password in cleartext then it would be easily copied. Even if the password would not be sent (e.g., OTP system) there is a *privacy* problem, which is different from confidentiality because if someone would sniff the traffic would know that someone is performing actions (e.g., "Lioy is

connected to the network"). There are no problems on sniffing the response (it just tells to the NAS if it is valid or not).

- **Fake AS response (to block valid or allow invalid user):**
  - This could be used because since this is UDP you could be faster in the answer then the real server. The first response received is considered valid, so the fake response could be used for DoS (to block a valid user) or give a positive answer even if the user is not valid. Response authentication is needed is such a case.
- **Changing AS response (Y → N or N → Y):**
  - It could be also possible to modify a response from valid to invalid and vice versa. For this reason, it is needed the authentication and the integrity for responses; response integrity is needed to avoid this kind of attack.
- **Replay of AS resp (if not properly tied to NAS req):**
  - A response could be replayed if the response is not tied to a specific request. If response is only "valid" or "invalid" it is possible to take the valid one and replay it to another user. In order to avoid this kind of attack, the response should contain the identification of the user (e.g. "Antonio Lioy which connected at XX:XX time is valid").
- **Passwords enumeration (from fake NAS):**
  - If someone can connect to the back-end network between NAS and Radius server, it could be possible to create a fake NAS and start sending requests to the Radius server. To avoid this, authentication from NAS requests is required.
- **DoS (many NAS requests from fake NAS):**
  - If it is possible to connect in the back-end network a fake NAS, it could be possible to flood the Radius Server with a lot of request, until it becomes unavailable. NAS are configured with a list of various Radius server. For this reason, if time expires and no response arrives, the NAS assumes that the server is busy and will switch to the next one. The resistance to the attack is proportional to the number of secondary servers that is possible to setup in the system.

RADIUS characteristics

For **data protection** there is packet integrity and authentication via **keyed-MD5:**

- Key is a shared secret (between NAS and Radius Server): inserting a fake NAS server in the network will result in the rejection of its requests;
- in case packets contain a password (e.g. user decided to use PAP and password was sent in clear) then the password is transmitted "encrypted" (not a real encryption, just obfuscated using MD5 that is not encrypting algorithm but just a digest one): $password \oplus md5(key + authenticator)$. This is padded with NULL bytes until a multiple of 128 bits is reached.

Radius supports all kinds of authentication (PAP, CHAP, token-card and EAP); Cisco provides a free server for CryptoCard; others supported also SecurID. The important thing is that Radius is an **extensible protocol** because each packet carries some attributes described with **TLV** form (***attribute type – length – value***). This would permit the introduction of new types without breaking compatibility.

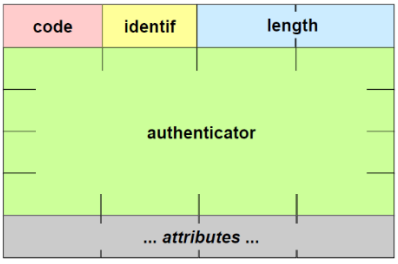


Figure 46 Format of a RADIUS packet

The format of a Radius packet is the following:

- *Code* (8 bits);
- *Identifier* (8 bits);
- *Length* (16 bits);
- *Authenticator* (128 bits);
- *TLV list* of attributes.

There are many packet types, some of them are:

- **ACCESS-REQUEST**: contains access credentials (e.g., username and password);
- **ACCESS-REJECT**: access is denied (e.g., due to bad username/password);

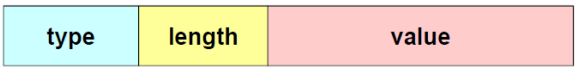
- **ACCESS-CHALLENGE**: requests additional information from the user (e.g., PIN, token code, secondary password);
- **ACCESS-ACCEPT** (*parameters*): access is granted, and some networks parameters are given (e.g., IPaddr, netmask, MTU, host, port, …).

Inside packets there is the **authenticator**, which has a double purpose: in the *server reply* provides *authentication* and *protection from replay*; *masks the password*. In Access-Request it is named *Request Authenticator* and it is 16 bytes randomly generated by the NAS. In the Server Responses, besides being named Response Authenticator, it is not random, but it is computed via a *keyed digest*:

$$MD5(code \,||\, ID \,||\, length \,||\, RequestAuth \,||\, attributes \,||\, secret)$$

Note that the presence of *RequestAuth* in the computation makes a response linked to a request, so that is not possible to perform replay attacks.

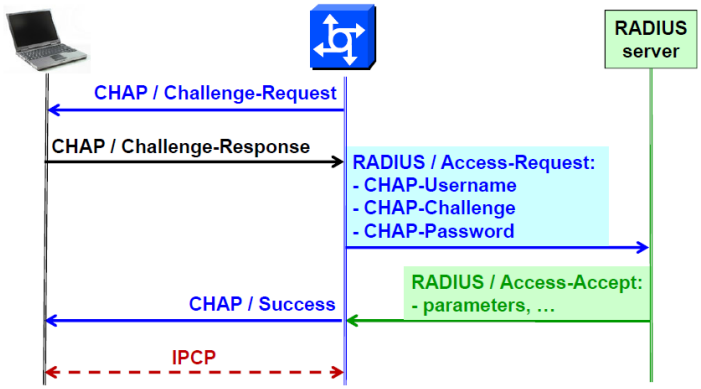
Attributes have TLV form and some of them are:



- **Type = 1 (User-Name)**
  - The value could be text string, network access identifier (NAI) or DN (Distinguished name)
- **Type = 2 (User-Password)**
  - The value is the *password ⊕ md5 (key || RequestAuth)*
- **Type = 3 (Chap-Password)**
  - The value is the user CHAP response (128 bit)
- **Type = 60 (CHAP-Challenge)**
  - The value is the challenge from the NAS to the user

The **NAI (Network Access Identifier)** is used to distinguish if the request is from a local user or if it is from a user belonging to a different security domain. NAI is in the form of username and in optionally at security realm. Rules says that all devices must support NAI up to 72 bytes long. The exact syntax for username and realm is detailed in the RFC but only ASCII characters < 128 are allowed, but all of them are allowed (even not printable characters). The username is the one used in the PPP authentication phase, that is the one used when opening the connection to the layer that does not necessarily match the application username.

Example: CHAP + RADIUS



CHAP is used for authentication and Radius to verify the authentication. On the left there is a user, in the middle there is the NAS and on the right the Radius server itself.

When user connects to the system the NAS is sending a CHAP packet containing a *challenge request*. The client enters the password and provides *challenge response*. Note that NAS does not have password and cannot verify if the response if valid or not. For this reason, it will create a packet *Radius Access Request* with all information: *CHAP-Username* is the one provided from user, *CHAP-Challenge* is the challenge that the NAS sent to the user and *CHAP-Password* is the response to the challenge. With this information the Radius Server can verify if the response to the challenge is good or not. Imagining that the response is good, the Radius Server will send a response *Radius Access Accept* with network parameters for the specific user. That will be translated to CHAP in the form of a packet *Success* and at that point the L3 will be enabled (e.g., IPCP). NAS is creating dialog with user willing to access the network and is transforming information from a protocol to another.

Radius assumes that we are inside the network access system of one single provider.

DIAMETER

Diameter is an evolution of RADIUS which puts special emphasis on roaming between ISPs. When visiting different infrastructures, DIAMETER permits authentication to another system. Many RFC have been defined:

- *RFC-3588 "Diameter base protocol"*
- ***RFC-3589 "Commands for the 3GPP"*** (ancestor of 3G, 4G, 5G)
- *RFC-3539 "AAA transport profile"*
- *RFC-4004 "Diameter mobile IPv4 application"*
- *RFC-4005 "Diameter network access server application"*
- *RFC-4006 "Diameter credit-control application"*
- *RFC-4072 "Diameter EAP application"*

Diameter has **better security than Radius**: it provides protection with an external mechanism on all packets exchanged. Rather than putting security inside the protocol, Diameter packets must be transmitted inside a secure network channel (such as IPsec and TLS). It is compulsory for Diameter client to support as a minimum IPsec and optionally also TLS. On the contrary, Diameter Server must support both security protocols.

From the point of view of the configurations, IPsec has got a format named *ESP* that provides encryption. IPsec must be configured with a **non-null algorithm** for both authentication and privacy. In case of TLS, it must support **mutual authentication,** which requires that the client must have a public key certificate.

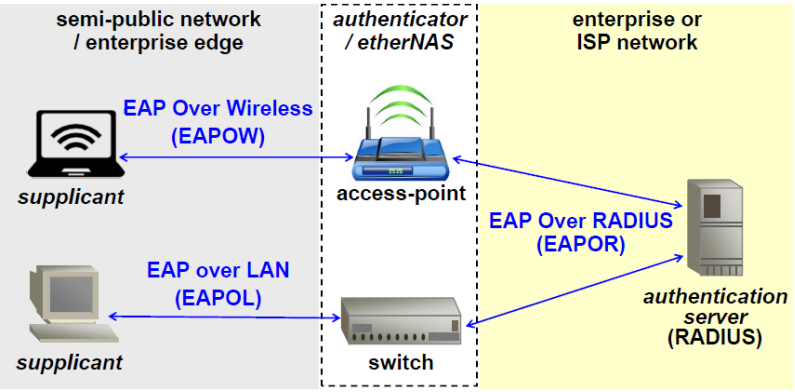
TLS has minimum requirements of supporting RSA for authentication, then RC4_128/3DES and MD5/SH1 for integrity. Optionally, it may support RSA with AES_128 and SHA1.

Radius and Diameter are both protocols that permit the centralization of access control.

IEEE 802.1x

Radius and Diameter have been used to define IEEE 802.1x, which is a more general architecture also known as **Port-Based Network Access Control**. It is an authentication architecture that works on L2. It verifies the identity and the authorization of users before letting them communicate. It could be *useful* in a wired network to block an unauthorized access, but it is **absolutely needed in wireless networks**. First implementations were immediately from Windows-XP and Cisco wireless access-points.

IEEE 802.1x is a framework, which means that it does not impose a specific authentication solution, but it supports many of them. Specially, it is a framework to perform **authentication** and **key-management**. The second point is needed for wireless networks, because the radio part of a wireless communication would be easily sniffed. With key-management is possible to define a shared symmetric key that can be used to encrypt traffic. It may derive session keys for use in packet authentication, integrity, and confidentiality. It uses standard algorithms for key derivation (e.g., TLS, SRP, …) and has optional security services (authentication or authentication and encryption).



On the left, the picture shows the **semi-public network** (or *enterprise edge*) *i.e.,* the network that contains the devices that want to access the network, which are called **supplicants**. The element which sits at the interface between the core network and the edge is named **authenticator** or **etherNAS** (it can be an *access point* or a *switch)* and, for supplicants, it is possible to connect in multiple ways to it. Particularly, 802.1x uses *EAP* to perform authentication and depending whether it is used on a wireless or a wired network, it is named respectively *EAP Over Wireless (***EAPOW***)* or *EAP over LAN (***EAPOL***)*. In any case, when

authenticator receives the EAP request from a supplicant, it will verify whether it is good or not by performing decapsulation and re-encapsulation of the packet in another protocol (Radius). On the back end there is *EAP Over Radius (***EAPOR***)* to check whether the user is a valid one or not.

802.1x - advantages

It exploits the application level for the actual implementation of the security mechanisms. There is a direct dialogue between supplicant and AS (authentication server), so the user devices are talking directly with the Radius Server. The network card (NIC) and NAS operate as "*pass-through device*" *i.e.*, they limit themselves to encapsulation and decapsulation. This is important because no changes are needed (at NIC and NAS) to implement new authentication mechanisms. These mechanisms must be implemented on the Radius Server and Supplicant only. This is important, because this security architecture does not need to change even if in future there will be an evolution in the authN techniques. Lastly, since it uses Radius, this system is perfectly integrated in AAA architectures that permit accounting too.

802.1x - messages

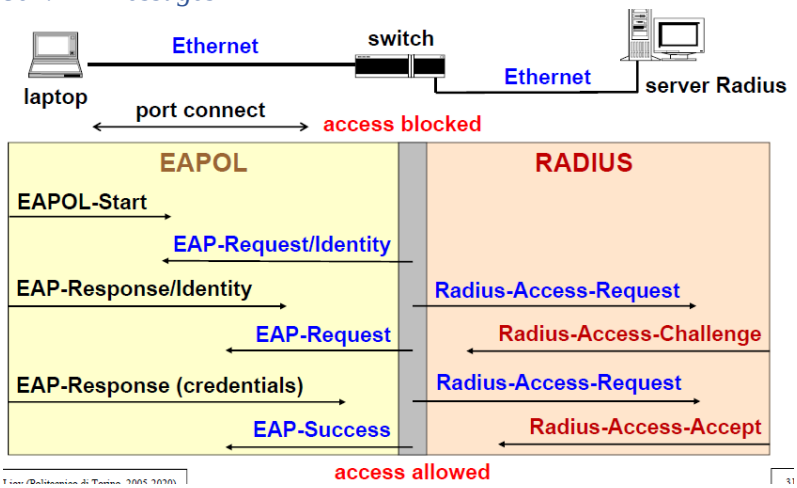


Figure 47 Authentication of a supplicant using 802.1x architecture: the switch is a pass-through device, it translates EAP messages into RADIUS and vice versa, but the connection is end-to-end

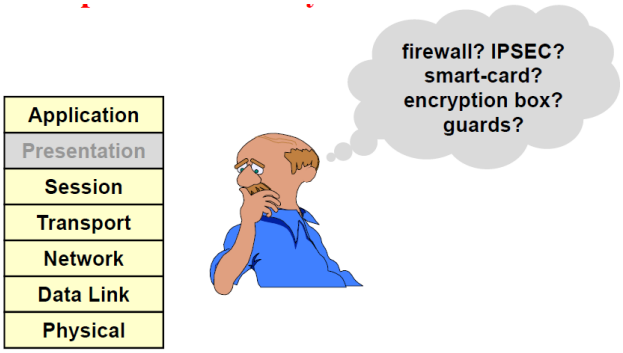
In the example the laptop is connected via Ethernet to a switch, in which by default all ethernet ports are locked. Then, the supplicant starts a negotiation by means of *EAPOL-Start*. The response from the switch is the *EAP-Request/Identity* and the Supplicant provides with the EAP-Response/Identity his identity, which is relayed to the Radius Server (*Radius-Access-Request*). Here the switch acts like a pass-through element. At this point, the Radius Server answer with a challenge via *Radius-Access-Challenge* packet which is then translated in an *EAP-Request* packet from the switch. The supplicant, again, will provide the response to the challenge which will be translated into another Radius Access Request. Then, the Radius Server will accept the user and the exchange will be concluded with an *EAP-Success* packet sent to the user. User will now be able to communicate with L3 and above.

Eduroam example

A major example of usage of Radius is **Eduroam** network. It is a worldwide network access control network which involves all universities and many research centers all over the world. Nearly one year ago there were connected 101 countries and other 18 countries under test. It uses 802.1x plus the Radius federation.

The Supplicant can see that there is an access-point in a university and can connect to it, because that connection is named Eduroam. The access point relates to the local AS, which is named **visit AS**. Since it will be using that Network Access Identifier (NAI) syntax, the supplicant will use his identifier (*e.g.*, s123456@studenti.polito.it) and the local Radius Server will know that it must go through the **Eduroam hierarchy** (national, international…) until it reaches the Radius AS in which the supplicant has created his credentials (*e.g.*, the PoliTO Radius Server), which is called **Home AS**. Once it has been found, there will be a direct connection through an E2E (End-to-end) virtual secure channel (*e.g.*, EAP-TTLS) between the supplicant and the Home AS to perform authN and then the latter will provide the answer to the access point, which will permit to the user to navigate.

Security implementation in OSI levels



The main question is "*Which is the best OSI level to implement security?*" with many possibilities and answers. Typically, "Presentation" is the only one in which security measures are not useful.

Unfortunately, there is not a single optimal level. The upper we go in the stack, the more specific can be our security function. For example, at application level is possible to identify the user, commands, and data. Security functions are also independent from the underlying network but, if the functions are placed at application level only, attacks at lower levels are possible (in particular, DoS attacks are available).

The more we go lower in the stack, the more quickly we can "expel" the intruders, but the fewer are the data for the decision (e.g., only the MAC or IP addresses, no user identification, no commands).

Shortly: there is not an optimal level. You can decide whether take one of these two risks or make a mixture by placing some security features at lower levels and then focus most of the security at application level.

When L3 is reached, one of the first thing activated is the DHCP, because access to network is given and now user needs to know the network parameters. DHCP is the protocol by which a device can ask to be assigned a valid network address. Unfortunately, the protocol is **non-authenticated** and a **broadcast** protocol which provides a response that carries *IP address, netmask, default gateway, local nameserver and local DNS suffix*. For this reason, the activation of a fake DHCP server is trivial, because the DHCP request is a L2 broadcast frame and the only thing an attacker needs to do is to stay in the same broadcast domain of the victim and sniff the DHCP request.

DHCP (in)security

Possible attacks from the fake DHCP are:

- **Denial-of-service**
  - This can be done by providing a wrong network configuration;
- **(Logical) MITM**
  - A valid IP address is provided to the victim, but it will be assigned a subnet with only the last two bits equal to zero. Therefore, only two addresses are valid: one of them is given to the user and the other to the attacker as his default gateway. In that way, the attacked machine is isolated in a subnet of its own (logically, not physically). In order to communicate with all the nodes in the world, the victim has to send everything through the attacker;
  - The replies could get the original node without passing through the attacker. For this reason, it is possible to activate NAT and it is possible to also intercept the replies.
- **Malicious name-address translation**
  - The attacker declares himself as local name server. Then, whenever the user needs to perform a name to address translation, the attacker will provide the wrong address. This is used, for example, for phishing and pharming.

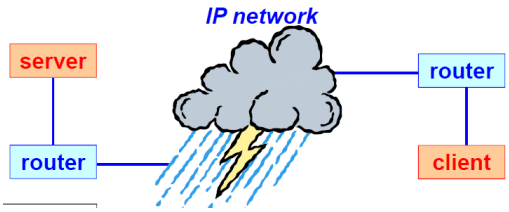
Various manufacturers have tried to provide some security improvement, such as switches (e.g., Cisco) that offer:

- **DHCP snooping**: accepts only replies from "trusted ports";
- **IP guard**: provides room only for IP addresses got from a valid DHCP server (but there is a limit on the number of recognized addresses).

There is also RFC-3118 "*Authentication for DHCP messages*" which uses *HMAC-MD5* to authenticate the messages, but it is rarely adopted because it is hardly configurable: since HMAC is a symmetric protocol, it is

needed to install a key on all the machines that need to use DHCP. This leads to the problem of key distribution. Furthermore, there is a problem of key management, because if a key is captured then it will be reusable and, since it is symmetric, any DHCP client could work as DHCP server too.

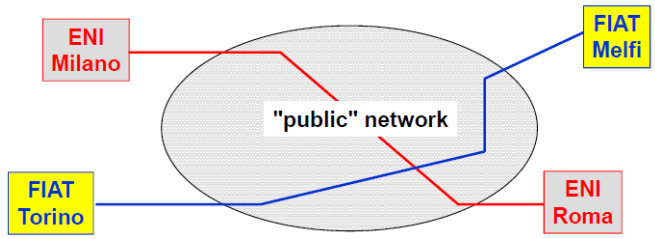
Security at network level (L3)



Layer 3, which given the today's use of Internet is IP, is a layer where is possible to create meaningful security features in general sense, because it is the first layer which provides end-to-end connectivity. This would permit to create both **end-to-end protection** for L3-homogeneous networks (such as IP networks) and **VPN (Virtual Private Network)**.

If it is possible to provide L3 end-to-end protection, so that data are secured as soon as they exit from the server/client and it does not matter whether the routers are not properly managed or if the network that is being crossed is unsecure, because data are protected as soon as they exit the network interface, until they reach the final network interface: therefore, the only possible attacks are those inside the client/server. Hence, security at L3 permits to forget about all the other attacks at network level (*apart from DoS*).

What is a VPN?



Virtual Private Network is a technique (hardware and/or software) that permits to create a private network while using shared (or anyway untrusted) channels and transmission devices. Rather than laying down their own cables and managing their own cables, there could be companies that would like to have a virtual segment of the network. For example, it is possible to say that FIAT packets are labeled as blue and they are allowed to be exchanged only between blue end point, while for ENI they are red and can be exchanged only through red end points. It is a good concept, but the devil is in the details: since it is not possible to picture packet blue or red and it is not possible to be sure on how they are switched, the implementation details matter.

There are three techniques to create a VPN:

- **Private addressing**
- **Protected routing (IP tunnel)**
- **Cryptographic protection of the network packets (secure IP tunnel)**

VPN via private addresses

In this very basic VPN implementation, the networks to be part of the VPN use non-public addresses so that they are unreachable from other networks (e.g., private IANA networks as those listed in RFC-1918). Hence, those networks are private in the sense that they do not require an authorization and the packets in this case are not globally routable. For example, Telecom provider that want to share its infrastructure with different customers could give a class of different addresses to each customer and then it could use access control lists on the routers in order to force packets to go only to the allowed destinations.

In several cases, this protection can be easily cracked if somebody:

- *Guesses or discovers the addresses*
  If a class of addresses is used from another customer and someone founds those class of addresses, it could switch its own address to infiltrate in that network.
- *Can sniff the packets during transmission*
  Since packets do not have any intrinsic protection, if it is possible to sniff the network, it also may be possible to read packets content.
- *Has access to the communication devices*

It is possible to read/change/inject whatever kind of packet.

There is no real protection here for packets, customers and even for the infrastructure maintainer. Therefore, the real level of security is close to zero even though this kind of service is commercially offered.

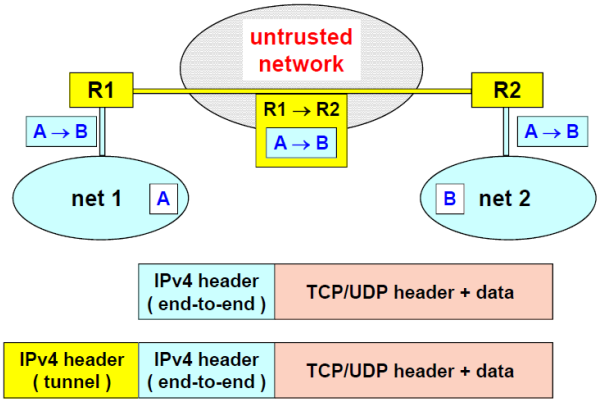
VPN via tunnel

This solution is a bit better than the previous one. Here routers encapsulate the whole L3 packet as a *payload* inside another packet. It can be **IP in IP, IP over MPLS** or other techniques. Before doing the encapsulation, the border routers perform access control to the VPN via **ACL** (Access Control List). For example, if a network belongs to *10.1* address, the destination can only be only someone else in *10.1*.

With this solution, providers obtain protection against malicious end users because it avoids that customers can change the subnet which they belong. However, this protection can be defeated by *anybody that manages a router* or can *sniff* the packets during transmission, for example it does not offer protection to customers against attacks that can be performed from within the geographical network (→ protection for providers but not for customers).

If we really want protection, we need to switch to other techniques.

VPN via IP tunnel



Network 1 and network 2 are of the same color because they belong to the same subnet. If an IP tunnel is being used, when the packets go from node A in subnet 1 to the node B in subnet 2, it reaches the border routers of subnet 1 which has the task to encapsulate it.

The router R1 will know that B is in the subnet 2 which is reachable through the border router R2 and will create another packet which goes from R1 to R2 that contains as payload the original packet.

In the picture it is shown the external IPv4 header of the tunnel. When packet will be received at router R2 it will be decapsulated and sent to the final destination. ***During transmission, the packet can be readable, manipulated, injected*** (again, no real security for the end user of the VPN).

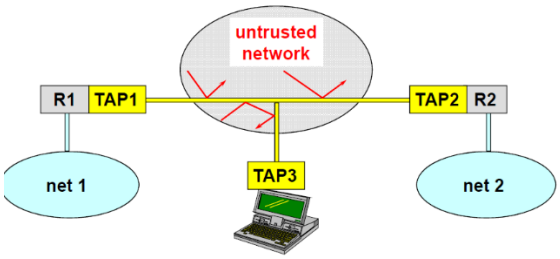
IP tunnel has also a problem of performance: **fragmentation**. If the packet has size equal to the MTU, then encapsulation will only be possible with fragmentation. In this case, the maximum performance loss is equal to 50%, because two packets are generated instead of one. The largest loss is for applications with large packets (typically the non-interactive applications, e.g., file transfer). This means that this solution can also be a performance killer.

VPN via secure IP tunnel

For the performance killer issue there is no solution but, with this last solution, something can be done for end users. Before encapsulation, the packets are protected with:

- **MAC** (*integrity and authentication*)
- **Encryption** (*confidentiality*)
- **Numbering** (*to avoid replay attacks*)

There is no digital signature, because it is **very slow** and would never be able to fit the speed of the current networks. If the selected cryptographic algorithms are strong, then the only possible attack is to stop the communications (*DoS*). Sometimes, this kind of VPN, it is also known as **S-VPN** (*Secure VPN*). This is the only VPN which is secure (→ beware with VPNs that are sponsored online)!



In the picture, there is a router and a **TAP** (*Tunnel Access Point*). The features are split: the router oversees the encapsulation/decapsulation while on the contrary the TAP is the one performing the cryptographic protection. If this solution is used and the TAP is given to be managed by an external network provider, then this is **fake security**. Because there should be two separated devices: the TAP should be managed by the client and the router should be managed by the ISP.

IPsec

IPsec is the IETF architecture for L3 security in IPv4/IPv6 to **create S-VPN over untrusted networks** and to **create end-to-end secure packet flows.** This is achieved by the definition of two specific packet types:

- **AH (Authentication Header):**
  To provide *integrity, authentication and protection against replay attacks*
- **ESP (Encapsulating Security Payload):**
  Provides *nearly the same functions provided by AH,* plus *(payload) confidentiality*

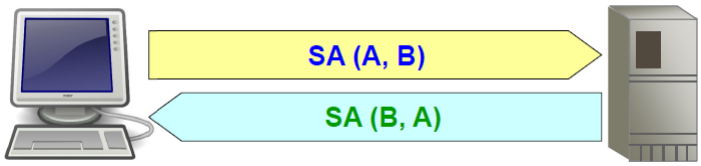
It is important to stress the fact that **confidentiality can only be provided for the payload**: it is never possible to encrypt the header, otherwise the intermediate systems would be unable to process the packets.

There is also a dedicated protocol for key exchange, named **IKE** (**Internet Key Exchange**), to create and distribute keys in IP networks.

The IPsec security services are:

- **Authentication of IP packets:**
  - *Data integrity*: the receiver can **detect** if the packet has been manipulated. It is **not** meant to avoid manipulation*;*
  - *Sender authentication*, that is a formal proof about the identity of the sender. Please, note that it does not correspond to an IP address: IP addresses must not be trusted, because they can be completely fake (*IP spoofing*);
  - *(Partial) protection against "replay" attacks*: there are issues with the fact that we are working at L3 and packets can be lost or duplicated;
- **Confidentiality of IP packets:**
  - *Data encryption* (for the payload only);

IPsec Security Association (SA)



These security features are associated to the concept of **Security Association (SA)**, which is a **unidirectional logic connection between two IPsec systems**. Each SA has associated different security services. To achieve full protection for a bidirectional packet flow between two nodes then **two SA are needed** (one from A to B and one for the packets from B to A). This means that, in theory, it is possible to have different security features and different algorithms for the two directions, but normally even if there are two distinct SA the same kind of protection/algorithms are used.

Imagine that the sender (node A) is trasmitting data (which means that it may want confidentiality) but the response from B is just a very simple thing (e.g. "received" or "bad") which does not need confidentiality: then it is possible to avoid the encryption of the returning packet.

Security Associations are managed through two local databases, **which are not real database** (*i.e.* no implementation of servers like SQL, Oracle, it means that is just a *collection of data*):

- **SPD** (**Security Policy Database**):
  - It contains a list of security policies to apply to the different packet flows;
  - A-priori configured (e.g. manually) or connected to an automatic system (e.g. ISPS, that stands for Internet Security Policy System);
- **SAD (SA Database):**
  - It is a runtime database that contains the list of active SA and their characteristics (*e.g.* algorithms, keys, parameters) to create protected traffic for that specific SA;

Suppose to be at a sending node within the TCP/IP stack. An IP packet has been created to be sent at L2, but on this node there's IPsec. When the packet is ready to be sent, the IPsec module starts working. The first question is: *which policy should be applied for this packet?* The answer is provided from the **SPD**. It can be "*you should apply this security rules*" or "*you don't need any kind of protection for this packet. Go straight to L2*". If the protection is needed and this is the first packet of this specific network flow, IPsec proceeds in creating a Security Association, otherwise there is already an existing SA and it proceeds in the reading of the parameter associated that SA by consulting the **SAD**. This will provide it the algorithms and parameters to enrich the packet and, finally, the IP packet will be protected with IPsec and sent to L2 for the actual transmission.

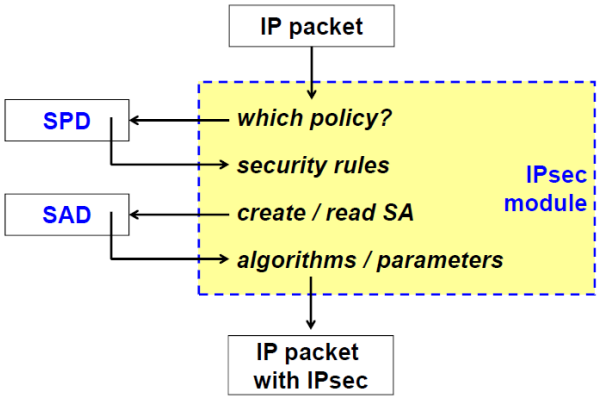


Figure 48 Logical operations performed by a IPsec module when sending a packet

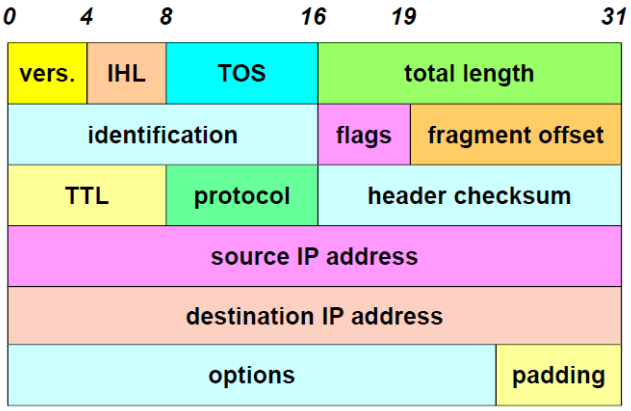


Figure 49 IPv4 header, as a reminder of the various fields: the value of some fields changes during the forwarding of the packet, and this must be taken into account when calculating the hash.

In the picture there is the IPv4 header. Since confidentiality can be applied only to the payload, is it possible to protect anything of the header? It could be possible to provide authentication and integrity. Typically those achievements require to compute a hash, which assumes that data are not changing though. Unfortunately, during the transmission some of these parameters change, such as the TTL and the checksum. Typically, NAT could not be used with IPsec, because by changing the source IP address, the hash changes as well and then the check performed at destination will fail. There are also some options (e.g. source routing i.e. the number of routers that should be traversed) which change.

IPsec ignores changing fields for creating hash in a syncronized way (the sender and the receiver must compute the same hash on the same parameters).

Transport mode IPsec

It is used for **end-to-end security**, that is used by hosts, not gateways (exception: traffic for the gateway itself, e.g., SNMP, ICMP). The original packet is cut in two parts and a new header is put between IPv4 header and TCP/UDP header. Thence, the IPv4 header will assert that it is transporting IPsec (instead of TCP/UDP) and then, inside the IPsec header, there will be another field that will tell what is being actually transported.

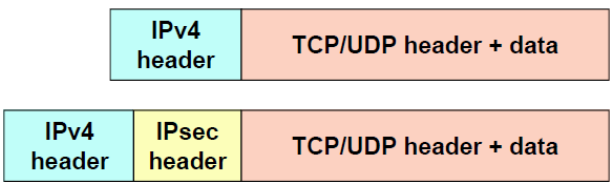


Figure 50 IPsec used for end-to-end security in transport mode

- **Pro**: *it is computationally light*
- **Con**: *no protection of header variable fields*

Tunnel mode IPsec

It is used to **create a VPN**, usually among gateways. It is not created among routers! The correct term is gateway: it is a contact point between a network which is assumed to be secure and a network which is not secure. The gateway adds protection by creating the secure IPsec tunnel.

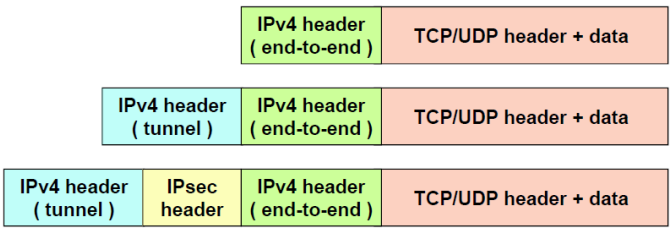


Figure 51 IPsec used in tunnel mode, to create a VPN. IPsec is not applied directly on the end-to-end packet, but on a tunnel between gateways

The gateway takes the original packet with the end-to-end header and puts it inside a tunnel which has as sender this gateway and has as destination the gateway which is protecting the network where the destination is located. With this IP in IP packet, the sending gateway will apply IPsec.

- **Pro**: *complete protection of the packet, header variable fields included*
- **Con**: *computationally heavy*.

Although not so common, IPsec in tunnel mode can be also used for end-to-end communication, even if it is usually adopted between border elements of the bigger networks: it is also called **site-to-site VPN**, since the entities are typically entire networks.

AH

AH stands for **Authentication Header**. There have been three major version of IPsec.

Mechanism of the first version (RFC-1826):

- *Data integrity and sender authentication*
- *Compulsory support of keyed-MD5 (RFC-1828)*
- *Optional support of keyed-SHA-1 (RFC-1852)*

Mechanism of the second version (RFC-2402):

- *Data integrity, sender authentication and (partial) protection from replay attacks*
- *HMAC-MD5-96*
- *HMAC-SHA-1-96*

The format of the header which is being added to the IPsec packet has:

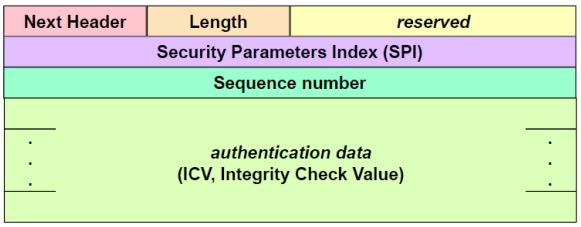


Figure 52 IPsec AH header format according to RFC-4302

- **Next header** field because this is a pseudo-protocol, so in the IP header there will be written that it is transporting AH, but then inside the AH there is the real transporting packet field;
- **Length** parameter of 1 byte to describe the length of the packet;
- **Reserved** bytes for future uses;
- **Security Parameters Index (SPI)**: 32 bits for referring in a quick and easy way to all the parameters that are needed to verify in the packet;
- **Sequence number** to avoid replay attacks;
- **Integrity Check Value (ICV)**: variable number of 4 bytes words to provide authentication data (digest).

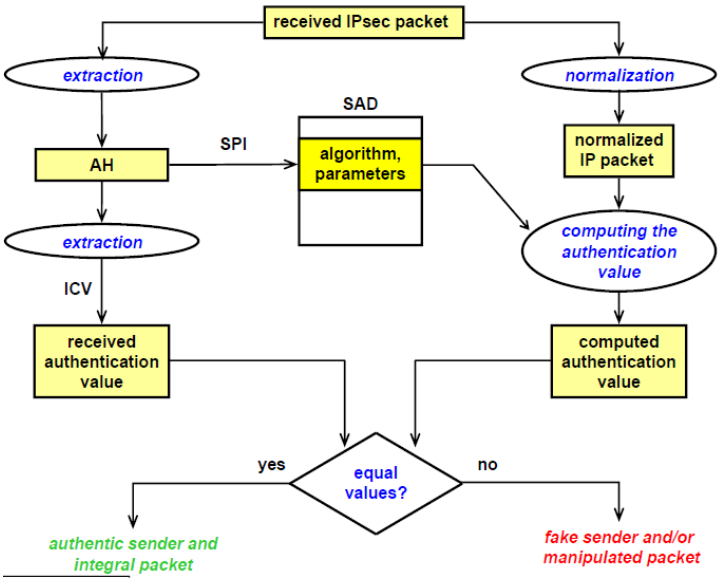


Figure 53 The processing of a received IPsec packet

When an IPsec packet is received it is protected with AH. It starts with the extraction of the AH and from it the **ICV** is extracted, which is the *received authentication value* (the digest computed by the sender). Then, on the received packet the **normalization** is performed, which means to put the packet in the same condition as it was at the sender in order to compute the same kind of hash. Once the normalized IP packet is available, it is needed to *compute the authentication value (ICV)*. For that, the *Security Parameter Index (SPI)* is being used inside the *Database of the Security Association* (*SAD*). It is a pointer that indicates algorithm and parameters to be used. These parameters can be used to compute the authentication value and then it is checked if the two values (the one computed and the one received from the sender) are the same. If the two values are equal, then the *sender is authentic, and the packet is integral*. If the two values are not equal, there could be a *fake sender and/or manipulated packet*.

An authentic sender is specified in the previous picture, but there is no place where the sender is authenticated. So, who is the sender authenticated here? The answer is in the fact that is being used a specific entry in the SAD. That entry negotiated with a specific node. For this reason, authentication is implicit in the process. The real authentication comes into play when we create the Security Association: this is the point in which the sender must prove its identity. Then the SA brings on that kind of authentication thanks to the usage of the correct algorithm/parameters.

Normalization for AH

The normalization is performed both at the sender and the receiver because the packet must be in the same state, otherwise the two ICVs would be different. It means that:

- The field named *TTL* in IPv4 and *Hop Limit* field in IPv6 must be reset to perform the computation.
- If the packet contains a Routing Header in the options, then:
  - Set the destination field to the address of the final destination;
  - Set the content of the routing header to the value that it will have at destination;
  - Set the Address Index field at the value that it will have at destination.
- Reset all options with the C bit (*change en route*) set.

The key digest is based on HMAC that has an additive parameter "96" at the end. The **HMAC-SHA1-96** is generated in the following way:

- Given **M** normalize it to generate **M'**
- Pad **M'** to a multiple of 160 bits (by adding 0x00 bytes) to generate **M'p**
- Compute the authentication base: **B = HMAC-SHA1 (K, M'p)** (where K is the key agreed with the sender)
- ICV = **96 leftmost bits of B**

Since we said that long digest is needed to avoid many kinds of attacks, the choice to get only 96 bits seems reckless: the reason is that here there is a conflict of interest between security and network managers. Network managers would need to have a header with a **fixed size**, in fact, if different algorithms are used for digest, they produce different lengths of digests, which is a problem for routers manufacturers (they would have a variable size header). It would be nearly impossible to process the header: obviously in this way there is lower resistance to the replay attack. In IPsec version 3 this aspect is enhanced.

ESP

If confidentiality is wanted, **Encapsulating Security Payload** (**ESP**) is needed. The first version (RFC-1827) provided only confidentiality. The base mechanism works on **DES-CBC** (RFC-1829), but other mechanisms are also possible. The second version also provided **authentication** but not for the IP header, so the coverage is not equivalent to that of AH. The advantage is that the **packet dimension is reduced,** and **one SA is saved**.

REMINDER: *IPsec is an architecture which is implemented with two kind of packets: AH or ESP, and they may be used at the same time*.

ESP can be used in transport mode and tunnel mode as well.

ESP in transport mode

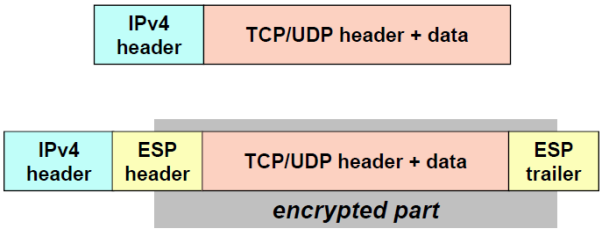


Figure 54 Usage of ESP header in transport mode

If used in transport mode, it is inserted between header and the payload. After a brief part in clear, all the rest will be encrypted up to a **trailer** which finishes it.

- **Pro**: the payload is hidden (including info needed for QoS, filtering, or intrusion detection)
- **Con**: the header remains in clear

ESP in tunnel mode

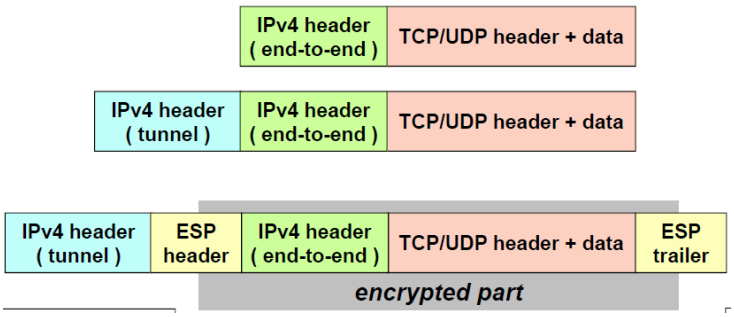


Figure 55 Usage of ESP header in tunnel mode

If it is used in tunnel mode, then first the tunnel is created, then the protection is applied to the tunnel payload. In this case everything of the original packet is encrypted, including the end-to-end header.

ESP – Format (II version)

It has in **clear** the *Security Parameters Index* (SPI) and the *Sequence number*. All the rest is encrypted data: it is necessary to have the entry corresponding to the SPI to read inside the encrypted data.

Let's assume that we have used **DES-CBC,** and this is the format:

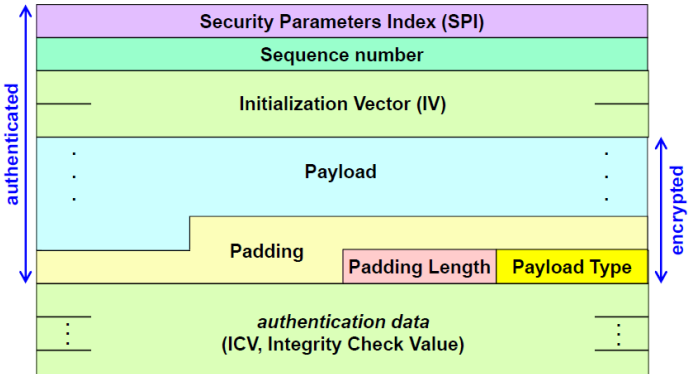


Figure 57 IPsec ESP-DES-CBC header format according to RFC-4306

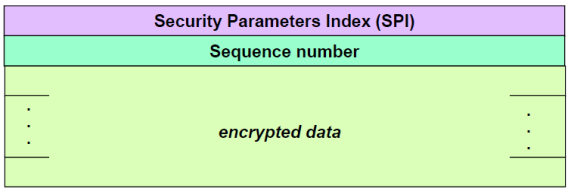


Figure 56 IPsec ESP header format according to RFC-2406

Since it is being used DES-CBC an *Initialization Vector (IV)* is required, which must be 64 bits. Then there is the payload itself. The part from the SPI to the padding (to reach a multiple of 64 bits) is the *authenticated* one. There must be also 1 byte to declare *Padding length* and 1 byte for the *Payload Type*, which is the layer 4 protocol that we are transporting. The fact that the Payload Type is encrypted is creating problems with network managers, because it means that if you have an IPsec packet using ESP, the intermediate

systems are **not** able to see the layer 4 protocol and they **cannot** perform QoS (or traffic differentiation), for example to give priority to TCP or UDP. If ESP is used not only for confidentiality but also for authN and integrity then, at the end there is a variable number of 4 bytes words containing the ICV (Integrity Check Value).

IPsec implementation details

Since there are many algorithms that can be used, RFC-4308 is defining two crypto-suites that anybody should implement for interoperability:

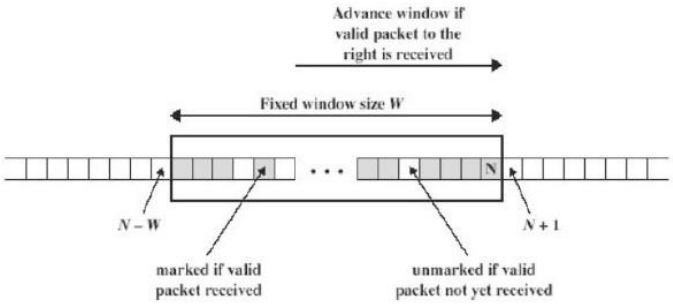
- **VPN-A** using ESP with 3DES-CBC and HMAC-SHA1-96: this is a kind of legacy VPN for compatibility with old systems.
- **VPN-B** using ESP with AES-128-CBC and AES-XCBC-MAC-96: it is the one used nowadays.

It is also possible to use **NULL algorithms** for ESP. It is possible to specify NULL for one of the two parts authentication or privacy, but **not simultaneously**. This permits to have some kind of "protection against performance" trade-off. For what concerns the sequence number, this is providing some partial protection against replay attacks and it works on a minimum window of 32 packets (64 suggested).

IPsec replay (partial) protection

When the sender is sending packets in sequence they are numbered. Then, since the network uses IP, the packets can be lost, duplicated, or can arrive out-of-order, which is a big issue. If the receiver has received packets 0-1-2 and then 4, is it possible to declare that packet 3 was cancelled by an attacker? Of course, it is not possible to answer because packet could be lost. Also, when the receiver has already received packet 53 and then it comes packet 3, is it possible to say that packet 3 is duplicated? No, because it could be an out-of-order packet.

To know if a packet, which in the sequence is before the last one that is received, is a duplicate the only way is to keep a list of the received packet and check if the packet was already received or not. The receiver cannot have a list of all the received packets, because that would be a huge number.



For this reason, a sliding window is used. There is a **fixed windows of size W** in which the packets that have been received are marked while the white packets are the one not received. When an old packet is received, if it is of a slot that has not been received it is accepted because that is an out-of-order packet. If it was already received it is discarded because it is a duplicated (either from the network or an attacker).

If an old packet (which is not inside the window) is received, there is no way to check. It is a big risk because if it accepted it could be a replay attack, while if it is discarded there could be a problem of communication.

If the traffic that is being protected is TCP, then accepting an old packet is not very risky because if it accepted but already received at the upper level it will be discarded because that segment will be already processed. On the contrary, if it is a UDP packet that should **never be accepted** and ask sender to send it again with a fresher sequence number.

The window proceeds, because if all packets to 20 are received and packet 21 comes, then the window moves of one packet always to the right. Every time a new packet with a sequence number bigger than the current one is received, the window slides. It **is not possible to wait** for the packets that are not received yet because they could be lost (and IP does not check retransmission).

IPsec v3

IPsec v3 changes paradigm about equal support to the two headers, in fact ESP is mandatory and AH optional: this means that it is possible to find implementations of IPsec that do not support AH i.e., integrity and authN for the payload only, not for the header. IPsec v3 has also added support for single source multicast. Moreover, since IPsec has been used mainly to create site-to-site VPN, which means channels with a lot of traffic, to avoid overflow of the sequence numbers IPsec v3 has introduced **ESN** (*Extended Sequence Number*), consisting of 64 bits even though inside packets there are still 32 bits only, because the 32 least significant are the ones that are transmitted: this is the default when using **IKEv2**. Additionally, rather than having the separated encryption of the payload plus the **MIC** (Message Integrity Code), it has been introduced the support for authenticated encryption (**AEAD**) and some clarifications about SA (Security Association) and SPI (Security Parameter Index) to get a faster lookup.

The algorithms used in IPsec are the following:

- For integrity and authentication:
  - (MAY) *HMAC-MD5-96*;
  - (MUST) *HMAC-SHA-1-96*;
  - (SHOULD+) *AES-XCBC-MAC-96*;
  - (MUST) *NULL* (only for ESP);
- For privacy:
  - (MUST) *NULL*;
  - (MUST– i.e. discouraged) *TripleDES-CBC*;
  - (SHOULD+) *AES-128-CBC*;
  - (SHOULD) *AES-CTR*;
  - (SHOULD NOT) *DES-CBC*.
- For authenticated encryption (AEAD mode):
  - *AES-CCM;*
  - *AES-CMAC;*
  - *ChaCha20 with Poly1305*.
- For authentication and integrity, it is possible to support longer digest:
  - *HMAC-SHA-256-128*
  - *HMAC-SHA-384-192*
  - *HMAC-SHA-512-256*

TFC (Traffic Flow Confidentiality) in IPsec v3

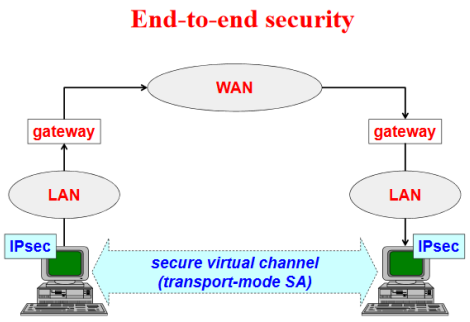
It is a padding in ESP that is put after the payload and before the normal padding: this is needed in order to not disclose what is the real size of the payload in the packet. The receiver must be able to compute the original size of the payload (e.g., possible with IP, UDP, and ICMP payloads thanks to the “length” field). For the same confidentiality reasons, IPsec v3 introduces the support for "**dummy packets**": this is in form of a nested pseudo-protocol (next header 59), and it is needed so that it is possible to keep transmitting even in absence of real data to send. Obviously, it makes sense to use dummy packets only if they are encrypted. The reasoning behind having dummy packets is that sometimes just the fact that peers are communicating can be a valuable information.

For instance, let us imagine that there is a general who wants to give an attack command, and that there are various squadrons: if the enemy could be able to see the general is communicating with one specific site, then he could suppose that in that site an action will be taken. Sending dummy packets to the other squadrons will make the messages indistinguishable for the enemy. Obviously, this reduces the global performances by increasing the data sent over the network, but it is a price needed to be paid to achieve this level of confidentiality.

Ways to use IPsec

End-to-end security

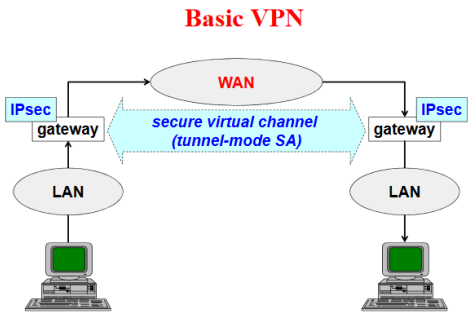
It consists in activating the IPsec module on the end nodes that are communicating. These nodes need a secure virtual channel, so they create a transport mode SA between them, so that the packets will be protected with the selected level just when they leave the network interface of the sender. In this way, there is no worry about LAN being insecure, or the gateway being managed by an untrusted party or the WAN being untrusted.



The great advantage is that security is implemented independently from the rest of the network: the only possible attack is DoS. The disadvantage is that **it is needed to install IPsec on both the machines** communicating: nowadays most operating systems support IPsec natively, but on some devices the module is not natively present, for example: mobile devices (Android and iOS) and embedded systems. So, the problem is basically that the kind of devices used must be considered, not only as it regards the availability of the IPsec software module, but also considering the computational power required to use it. Another critical aspect is about security management: if IPsec is used to protect all the computers in a big LAN, then an efficient system for managing the IPsec configurations on all those nodes is needed. Finally, if for the secure virtual channel ESP is adopted with non-null encryption algorithm, then the traffic cannot be sniffed not even from inside the LAN, so for example adopting an IDS (*Intrusion Detection System*) inside any of the two LANs is impossible, because one would not see the real content of what is transported: in this case it is needed the IDS to be placed directly onto the nodes, rather than in the network.

Basic VPN

The IPsec modules are placed directly on the gateways that are protecting the internal network from the external one. In this case, since the channel must protect all the traffic between the two networks, it must be a tunnel-mode SA, because packets coming from one network to another need to be encapsulated.

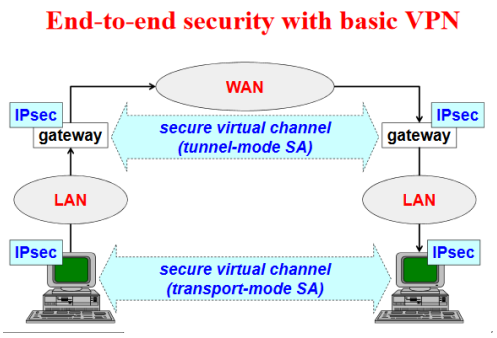


The main assumption on this architecture is that the internal network is secure and trusted, so that the only concern comes from attacks over the WAN. It means leaving an open door for internal attacks and it is not possible anymore to perform authentication of the real endpoints of the communication: only the intermediate elements, that are the gateway, are authenticated. This model is also called **site-to-site VPN**.

In this schema, usually gateways do not suffer the problem of having IPsec capabilities, but nevertheless there is a problem of computational capacity: since the gateways must perform the tunnelling for all communications, they could be *overloaded*. Typically, in this case the gateways are equipped with some powerful CPU or hardware accelerators, such as an HSM**.** On the other hand, management is greatly simplified, because it is not needed to manage the configurations for all the end nodes, but only on the gateways. Since there is not end-to-end security among peers, there is also the possibility to inspect the internal traffic.

End-to-end security with basic VPN

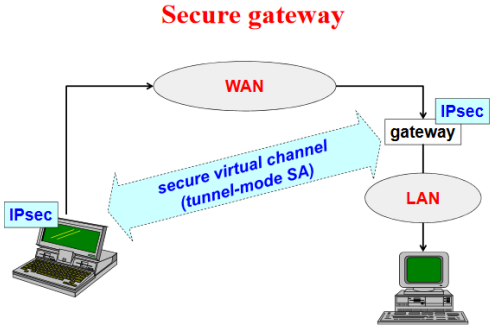
It is an example of the principle of "*defence in depth*", that is creating more than one defence line. Here there is a double activation of IPsec: between end nodes and between the gateways. In this way it is possible to have two defence lines or also to balance the security: for example, the end-to-end connection, that is the transport mode SA, could be used for authentication and integrity only (e.g., to clearly identify who the sender is), while encryption could be activated only between the gateways to protect from sniffing in the WAN and maintain the possibility to inspect the traffic on the LAN.



The problem is the management of the whole structure, that is the management of the maximum number of systems (like in the first architecture) and all the gateways (like in the second one).

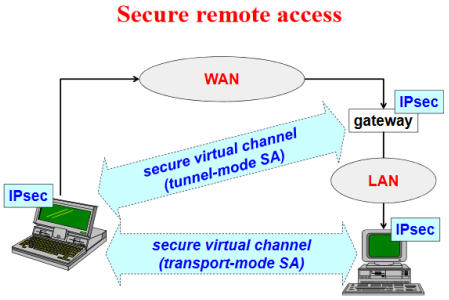
Secure gateway

The concept of secure gateway is that there are mobile users, for example employees who travel and need to connect to the internal company network. In this case, it is possible to deploy IPsec on the mobile device of the user and create a secure virtual channel in tunnel mode SA between the device and the company gateway. This permits to have all the traffic from the user to any internal server in the company network protected; additionally, in this way the gateway will also be able to perform authentication and authorization.



Secure remote access

In this case there is again the possibility of a double defence line: there is tunnel mode between the mobile node and the gateway plus an end-to-end virtual channel between the mobile node and the final destination. In this case typically the tunnel mode is used only for authentication and authorization (to give access to the internal network), while the transport mode SA is typically used for end-to-end protection, according to the kind of protection needed.



IPsec key management

It is an important component which provides to all the parties the symmetric keys used for packet authentication and eventually encryption. To distribute these keys, it is possible to use an OOB (out-of-band) approach, for example passing the keys **manually**. It is physically possible if there is a limited number of nodes and there is the ability to directly inject the keys to those nodes. With many nodes some **automatic in-band** key distribution is needed but this in turn needs a protocol.

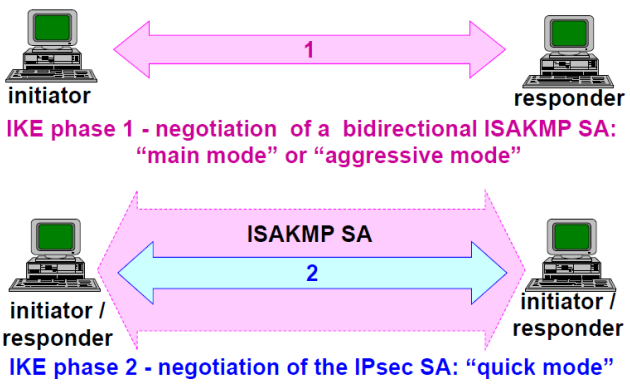
ISAKMP

An integral part of the IPsec architecture is the protocol **ISAKMP** (***I**nternet **S**ecurity **A**ssociation and **K**ey **M**anagement **P**rotocol)*. Described in RFC-2408, it contains the *procedures needed to negotiate, set-up, modify and delete a SA*. Unfortunately, there are no information about the key: the key exchange method is not fixed, and it can be an OOB one (and ISAKMP is used just to select the proper key) or an in-band method by using **OAKLEY, a protocol for authenticated exchange of symmetric keys.**

IKE

The combination of *ISAKMP* and *OAKLEY*, widely adopted, has been renamed **IKE** (***I**nternet **K**ey **E**xchange*). It is one of the most complex security protocols because it works in the following way:

- First there is the creation of a SA to protect the ISAKMP exchange;
- The created SA is used to protect the negotiation of the SA needed by IPsec traffic;
- The same ISAKMP SA may be reused several times to negotiate other IPsec SA.



On the left there is a possible schema of IKE operations. In the phase one there is a node named the **initiator** because it takes the initiative to open the IPsec channel towards another machine named **responder**. In IKE phase 1 it is possible to negotiate a bidirectional ISAKMP SA to protect traffic in both directions and it is possible to choose between *main mode* and *aggressive mode*.

Once the SA is in place, then any of the two nodes can act as initiator to create an IPsec SA. The phase two can be negotiated using the *quick mode* because all the traffic is already protected by the SA. The main modes of operation are:

- **Main Mode**
  - It requires the exchange of *6 messages* (→ quite slow)
  - *Protects the parties identities:* IP addresses are of course always visible to an attacker but remember that in IPsec the authentication is given by what has been used during the authentication phase to create the key. Those "cryptographic" identities are not disclosed.
- **Aggressive Mode**
  - *3 messages (but does not protect the parties identities)*
- **Quick Mode**
  - *3 messages*
  - *Negotiation only of the IPsec SA*
- **New Group Mode:** used to communicate to the other peer to inform it about a change in the algorithm or the key that is being used to protect the traffic.
  - *2 messages* only

When a SA is opened the authentication is needed; there are four ways to perform authentication:

- **Digital Signature**
  - *Non-repudiation of the IKE negotiation, which means it is not possible to deny the request to open the secure channel.*
- **Public Key Encryption**
  - *Identity protection provided in the aggressive mode*
- **Revised Public Key Encryption**
  - *Less computationally expensive, only 2 public-key operations*
- **Pre-Shared Key**
  - *The party ID may only be its own IP address (which creates problems with mobile users)*

VPN concentrator

IPsec nowadays is mostly used to create site-to-site VPN. This means that there is the problem of the performance of gateways. For that reason, it has been proposed the existence of a **VPN concentrator** which is a **special-purpose hardware appliance that acts as a terminator of IPsec tunnel** which is used for *remote access of single clients* or to *create site-to-site VPN*. Since it is implemented in hardware, it has very high performance with respect to the low costs. It should be considered in a company/campus scenario in which a lot of traffic is exchanged, and many people remotely do their job.

IPsec conclusions

IPsec can be applied **only to unicast packets** (*no broadcast, no multicast, no anycast*) because it is needed to identify the peer and exchange a key; as we have seen, IPsec v3 has an addition for single source multicast, but this is an exception. Moreover, since the reciprocal authentication is needed, IPsec **applies between parties that activated a SA** by *shared key*s or by *X.509 certificates*. In general, IPsec is good for **"closed" groups**: IPsec cannot be used for example to create e-commerce service (because users of the service are unknown: no information about keys or certificates).

Basically, IPsec is providing some security to the upper-layer traffic, which is carried inside IP packets. However, there are many protocols carried over IP that inherit the IP insecurity since **IP addresses are not authenticated,** and **packets are not protected** (if IPsec is not used). For these reasons, all protocols that use IP as a carrier can be attacked. Normally, the attacker tends to go to those protocols that are neglected, the so-called "service" protocols (i.e., the non-application ones, such as ICMP, IGMP, DNS, RIP and so on).

"Service" protocols security

ICMP security

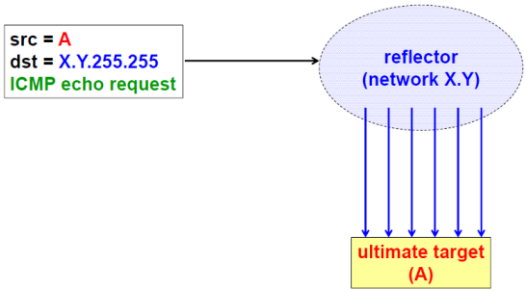
ICMP is the *Internet Control and Management Protocol*. It is vital for network management and for this reason many attacks are possible because it has **no authentication**. Looking at the ICMP functions there are:

- *ICMP echo request/reply*
  - Ping flooding or Ping bombing
- *Destination unreachable (network/host/protocol/port unreachable)*
  - Since packets are not authenticated, fake nodes could send destination unreachable to the sender, the latter will close the communication thinking that the destination is truly unreachable: this is a DoS.
- *Source quench (deprecated with RFC-6633, year 2012):*
  - It was meant to provide a mechanism for congestion control: senders *must* react to ICMP Source Quench messages by slowing transmission on the connection, because intermediate buffers are full and intermediate nodes are unable to send data to destination at the required rate.
  - Again, these ICMP Source Quench Messages have no authentication, which means that fake packets could be sent with the aim to slow the destination down (DoS).
- *Redirect:* it is an ICMP message that can be sent by an intermediate node when it detects that the packet has taken a wrong path
  - It makes possible to create a logical MITM: an attacker can redirect the sender to a malicious node under its control and there it can perform the MITM attack;
- *Time exceeded for a datagram*: it is an ICMP message normally sent by an intermediate node when it is processing a packet with TTL=0
  - Receiving this message usually means that there are loops in the routing plan, so the host closes the connection.
  - Faking this message can cause DoS for the sender.

It is not possible to protect ICMP with IPsec and these kinds of problems are always possible. The only possible thing is to cooperate with the network manager and to keep an eye on all the ICMP packets that goes through the network.

Smurfing attack

ICMP can also be used to perform the *Smurfing* attack (a type of DoS attack). There is a target (A) which is attacked by creating an ICMP echo request (Ping) with a fake sender, like if the ping was sent by the victim. The destination is the broadcast address of a whole network. There is a ping to all the nodes inside that network, which is named **reflector**, because it is sending back a *pong* (*ICMP echo reply*) for this ping. Since it is addressed to the broadcast address, all the active nodes inside the network will answer back. With a single ping packet, it is possible to reach 60k replies. The final target will be overwhelmed with messages, while the reflector network will be quite busy because there will be a lot of ICMP traffic to forward.



To prevent this attack, typically for external attacks the network **reject** IP broadcast packets at the border. Imagine a border router with *serial0* interface in the external network. It is possible to declare:

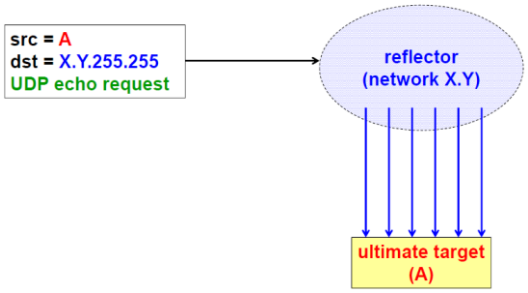
```
interface serial0
no ip directed-broadcast
```

In this way all broadcasted IP packets will be discarded. But this is a solution for external attacks. The network is still sensitive for an attack coming from within the intranet (broadcast in LAN cannot be disabled because it is required by protocols). In this last case, the only way to counter an internal source of smurfing is to identify the attacker via Network Management Tools and then try to stop its machine.

Fraggle attack

Fraggle is an attack quite like smurfing but it is performed using UDP, rather than ICMP. The philosophy is the same: a fake node sends an *UDP echo request*.

These requests are nowadays disabled by default. Therefore, this success of this kind of attack depends on the number of clients inside the reflector network that have the *UDP echo request* enabled.

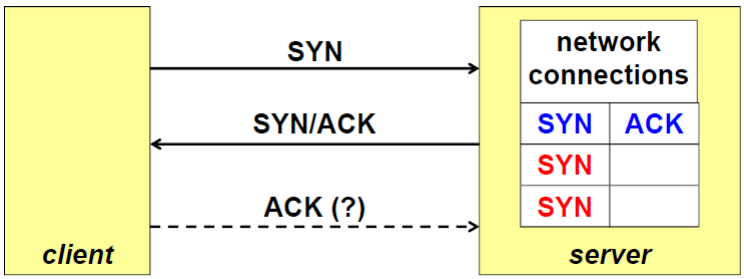


ARP poisoning

ARP is the Address Resolution Protocol used to discover the Level-2 address exploiting the Level-3 address knowledge. The result is then cached inside the ARP table. The **ARP poisoning** is performed like this:

- Nodes accept ARP reply even if they did not send an ARP request, because it could avoid the need to send an ARP request in the future when contacting that node;
- Thus, it is possible to send unsolicited wrong ARP Replies to nodes;
- Nodes will overwrite static ARP entries with the dynamic ones (obtained from ARP replies)
    - *Most stacks do not check that the source HW address inside the ARP field is coincident with the source field in the 802.3 packet*
- This is used by many attack tools (e.g., Ettercap)

TCP SYN flooding



Remembering the three-way handshake to open a TCP connection, a first packet is sent containing the *SYN* flag. The receiver will answer back with a segment with both *ACK* and *SYN* flags. Normally the client would close/open the channel with a final *ACK*.

The client can be a malicious one and send a *SYN* but never send the final *ACK* back. The server goes further to another line on its table of connections and the client goes back to the first step and send again another *SYN*. At some point the table of connections will be full (maximum size reached) and it will not be able to open new connections for real users. That will be a DoS for clients who wish to connect to the server.

Note that the final missing ACK can be also due to a network problem. In TCP-IP documents it is written that after a value of seconds (typically 75s) the server will check its connection table and if half connections are opened, those will be reset.

To protect against SYN flooding it is possible to:

- **Decrease the timeout**
  - *There is the risk to delete requests from valid but slow clients*
- **Increase the table size**
  - *It can be circumvented by sending more requests*
- **Use a router, in front of the server, as "SYN interceptor"**
  - *It substitutes the server in the first phase*
  - *If the handshake completes successfully, then transfers the channel to the server*
  - *"aggressive" timeout (risky!)*
- **Use a router as "SYN monitor"**
  - *It kills the pending connection requests (RST)*

If the network manager notices a huge quantity of SYN (more than the usual amount) that should be a right time to start investigating.

SYN cookie

Nowadays, this is the best solution against SYN flooding. It is an idea of D.J. Bernstein which noticed that the attack is possible because, when the server is receiving a SYN, it is storing something in a table about that client. The idea is not keeping the state of connection inside the server but inside the client instead. The idea uses the TCP sequence number of the SYN-ACK packet to transmit a cookie (keyed-digest) to the client and later recognise clients that already sent the SYN, without storing any info about them on the server. This is possible because a client answers back with an ACK which has the sequence number + 1 (which means the keyed-digest + 1) and the server will check if the keyed-digest is a valid one made before by itself. More in detail, the server initial sequence number is built as follows:

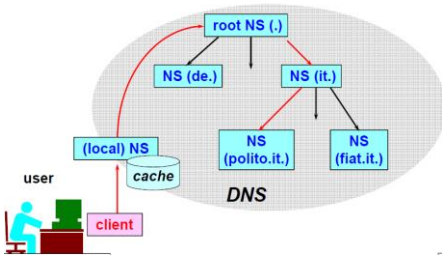
- let **t** be a slowly incrementing timestamp; then $\boldsymbol{t}\ mod\ 32$ gives the top 5 bits of the sequence number;
- let **m** be the maximum segment size (MSS) value that the server would have stored in the SYN queue entry;
- let **s** be the result of a cryptographic hash function computed over the server IP address and port number, the client IP address and port number, and the value t. The returned value s must be a 24-bit value.

The server, after receiving the ACK and retrieving the keyed digest:

- Checks the value **t** against the current time to see if the connection has expired.

- Recomputes **s** to determine whether this is, indeed, a valid SYN cookie.
- Decodes the value **m** from the 3-bit encoding in the SYN cookie, which it then can use to reconstruct the SYN queue entry.

DNS security



The **DNS** is the ***D**omain **N**ame **S**ystem*, which provides translation from names to addresses, but it also does the reverse job. It is a vital service because every user uses names instead of IP addresses to access websites. It works by means of **queries,** performed over port *53/UDP*, and **zone transfers** (transfers of information between servers) over port *53/TCP*. By itself, it has **no intrinsic security**: *DNS-SEC* is under development and it has not been implemented yet.

The DNS architecture is the one in the picture above. It is a hierarchical disposition of servers, starting from the one that is local (which means inside the network of the client). If the user wants to access to a website, the local DNS will check inside its caches. In case of *cache miss*, the local DNS will interrogate the *root NS*, which typically has not the answer, because the root manages only top-level domains known as dots (“.”). The question then will be redirected from the root NS to the right domain server (first to *it.*, then e.g., to the *polito.it*) to get the answer.

The picture above depicts the logical path to take to recover the IP address, but actually the name server is always responding to the local DNS, which typically saves in cache the new answered queries.

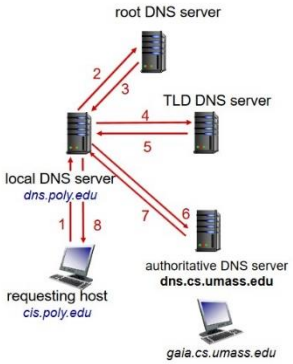
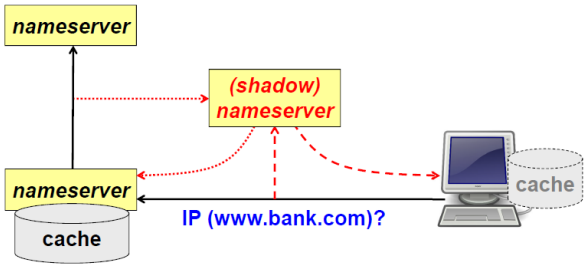


Figure 58 DNS name resolution with iterated query

DNS shadow server



The first kind of attack is a shadow/fake DNS server. If someone can intercept and sniff the request to the local DNS, it can possibly provide a fake response too, in order to give a fake IP address to the victim.

On the contrary, if someone can intercept the request going from the Local DNS towards the Root DNS, it is possible to provide the fake response to the local DNS, which will give the fake IP address to all the clients of that network that will provide that DNS query.

Microsoft implemented on Windows a cache for DNS, which is violating the general rules for DNS. In this case, an attack of this kind is much more effective because cache on client is not often updated.

DNS cache poisoning

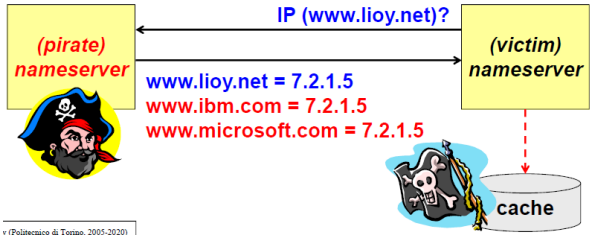


Figure 59 DNS cache poisoning by setting up a (pirate) nameserver and providing wrong answers

DNS has cache to avoid repeating questions. In this kind of attack a domain is created (*www.lioy.net*). Every time a domain is created a *nameserver* is also needed. In this case it is a pirate-nameserver because every time someone asks for the nameserver (e.g., the address of www.lioy.net) the pirate nameserver will answer with the correct one plus inserting also other wrong information. If the victim nameserver is not correctly programmed and configured, it will accept the additional answers and overwrite (if already stored) the information available inside its own cache.

The problem of this attack is that the victim must request that specific pirate Address resolution. For this reason, it is possible to perform another version of cache poisoning.

DNS cache poisoning (2nd version)

In this version the attacker has a pirate DNS client and asks to the victim recursive nameserver the resolution of the address. Since the search would take time, the attacker sends the answer with a fake source address, as if the address comes from the authoritative NS.

This attack is much more effective since it is not necessary for the victim to be tricked into requesting the resolution for the pirate's nameserver.

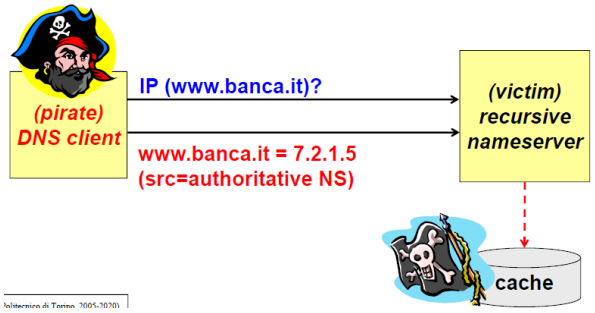
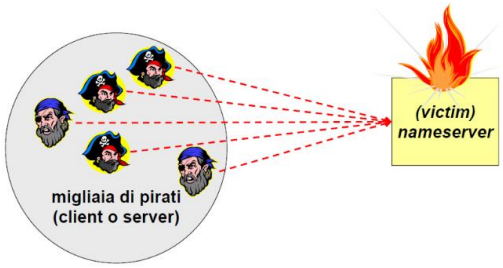


Figure 60 DNS cache poisoning by setting up a DNS client and providing question and wrong answer with IP spoofing

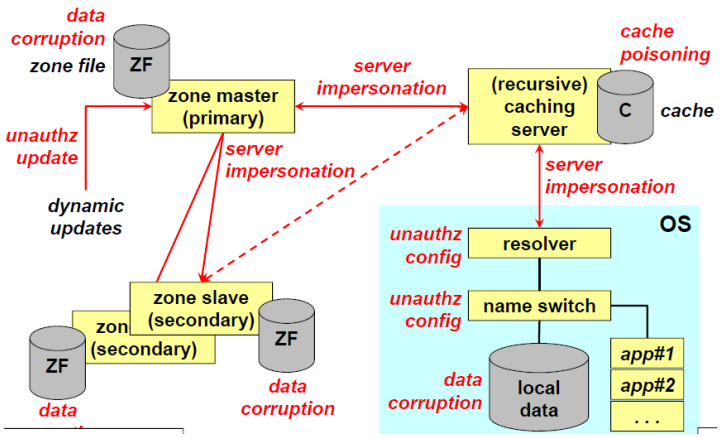
(DNS) flash crowd



The only possible way for a DoS attack to DNS is to have a lot of clients which ask questions to a nameserver. The DNS is the target of many attacks, because if the nameserver is blocked, all the domains related to that server will become unreachable.

Name-address translation

Of course, name-address translation request starts from a device running an operative system. Inside the OS various applications need to resolve names by getting the corresponding address. To do so, the OS exploits a component named **name switch**, which can decide which service is being used to switch from a name to the corresponding address. For example, the *etc/hosts* file which is a local data. If someone is able to add a name-address pair within that file, it is possible to force applications to connect to a fake server.



The name switch typically decides the priority between the internal file and some external service. The most common external service is the one approached by the resolver, that is a component which implements the DNS client and can talk with the DNS infrastructure.

In case the name is not found inside the local data, then the resolver will connect to an external service (typically a caching server). The caching server can be eventually a recursive one, which means that is taking care by itself of putting all the questions. The caching server will contact lot of servers, according to the hierarchy described before, which connect to a specific zone. Each zone is managed by a master (*primary name server*) and many slaves (*secondary name servers*) which maintain a database named *zone file*, which contains the correspondence between names and addresses. The caching server may contact the primary server or the secondary servers (in case the primary is not available or overloaded).

Normally, the zone master is updated manually by the administrator. Unfortunately, Microsoft added the possibility to perform dynamic updates directly by the clients. When a Microsoft network is joined and an address is assigned, it is possible to inform the DNS master about the IP address received: this is quite dangerous because it permits an uncontrolled modification of DNS data.

Possible attacks to this infrastructure are:

- **Data corruption to the local data**: if an attacker gets access (e.g., through a keyboard or a malware) to a client, it can try to insert among its local data some information to produce a wrong mapping;
- **Unauthorized configuration** of name switch/resolver (rather than pointing to the actual external name server, it is possible to point to a fake one which will provide wrong answers);
- In the communication between the resolver and server and between caching server and zone server there can be **server impersonation** (i.e., someone posing as the real server), because DNS does not have any kind of authentication;
- Every time there is a cache, there could also be **cache poisoning**;
- **Data corruption** may happen also on the zone file if the attacker can penetrate the zone master;
- Dynamic **updates** could be performed by **unauthorized** nodes;
- Since the master sends to the secondary servers a copy of the zone file, again there could be a **server impersonation,** which means that a fake master could send to the secondary servers a wrong copy.

As seen, there are many local attacks (such as data corruption), and some are network attacks basically due the lack of authentication.

DNSsec is needed

In February 2008 Dan Kaminsky, a researcher, found a new attack that makes cache poisoning simpler to execute, more difficult to avoid and applicable also to the 1st level NS records (e.g., `com`, `it`). Then in July 2008 the first advisories and patches came out and then there was a talk by Kaminsky at Black Hat. In September 2008 USA made compulsory use of DNSsec for the `.gov` domain starting from January 2009.

Today, there is still no protection on `.it` domains (as many other countries in the world).

DNSsec

DNSsec is basically the digital signature of DNS records which makes impossible to create fake answers. Even if a record is digitally signed, there is another problem: *who is "authoritative" for a certain domain?* For example, if you get a grade from Prof. Lioy then it is legitimate, because he has the right to sign grades but can put digital signatures only on grades of Lioy's courses. Digital signature is not enough, an infrastructure is needed to tell who is authoritative and who has the right to digitally sign an answer. Each digital signature also requires a certificate: *which is the PKI?* (certificates, trusted root CA).

The problem is even bigger because many servers all around the world have a complex management of the DNS infrastructure. For this reason, it is needed:

- *Hierarchical and delegated signatures*
- *Distributed signatures*

Another important problem is about the handling of non-existent names:

- *The ABSENCE of a record must be signed too, which is difficult*
- *This requires sorting of the records*

Some issues with DNSsec

- **The signature is available only on the answers and not on queries**, so it is not possible to know who put the question;
- **There is no root CA**, but the level 1 keys are distributed OOB, which makes nodes possible target of an attack if keys can be modified;

- **There is no security in the dialogue between the DNS client and DNS (local) server**
  - Use IPsec, TSIG or SIG(0), that have been proposed but not implemented;
- **Signature to be performed by the DNS server lead to:**
  - Computational overhead;
  - Management overhead (on-line secure crypto host is required)
- **Bigger record size,** due to the presence of the signature and the fact that it is needed also to attach the certificate;
- **Scarce experimental results** about what is the correct configuration and what is the actual loss in performance;

Routing security

Normally, there is low security in the system **access to routers** for management, for example *telnet* or *SNMP (Simple Network Management Protocol*). In addition to routers management, there is low security in the **exchange of routing tables**:

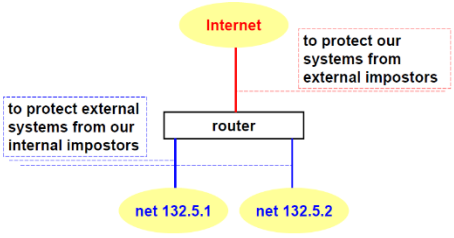
- Authentication based on IP addresses. This can be exploited by an attacker to insert fake routing tables;
- Optional protection with a keyed-digest is typically not implemented because:
  - A shared key is required! However, since it is shared, if it is distributed to too many routers then they can impersonate each other;
  - Key management is required! Since the key should be distributed to many routers OOB.

Finally, the redirect attack on ICMP will permit variations on dynamical routing on end-nodes.

Protection from IP spoofing

It is not possible to prevent a host from changing its own address, but it is possible to try to limit the diffusion of IP spoofing. There are some RFCs that suggest to network managers how to increase the protection from IP spoofing, both to protect a host from external impostors and to protect the external world from impostors internal to the local network.

- RFC-2827 "*Network ingress filtering: defeating Denial of Service attacks which employ IP source address spoofing*"
- RFC-3704 "*Ingress filtering for multihomed networks*"
- RFC-3013 "*Recommended Internet Service Provider security services and procedures*"



Looking at the picture, it is possible to notice two networks: *132.5.1* and *132.5.2* (they are 2 subnets connected to a border router connected to internet). On the red line, which connects the router to internet, it is wanted protection for the system from external impostors. Typically, from internet is possible to receive any address with one notable exception: it is not possible to receive any address that belongs to the internal networks (not *132.5.1* and *132.5.2* addresses). On the connections between router and subnets, instead, it is possible to receive only packets that belong to the specific network.

By placing filters on the lines, it is possible to protect against both external and internal impostors. If all providers would apply this simple rule it would be possible to avoid IP spoofing globally.

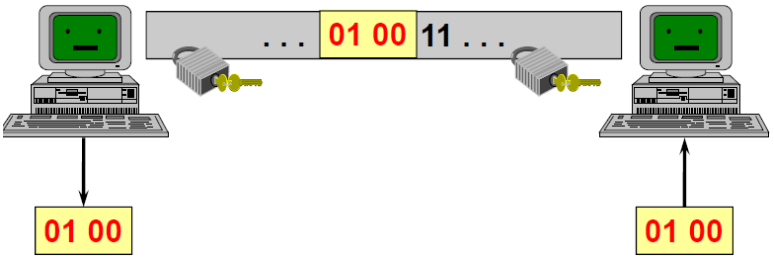
Possible syntax (CISCO language) in which is possible to apply filters is the one in the figure on the right.

```
access-list 101 deny ip
  132.5.0.0 0.0.255.255 0.0.0.0 255.255.255.255
interface serial 0
ip access-group 101 in

access-list 102 permit ip
  132.5.1.0 0.0.0.255 0.0.0.0 255.255.255.255
interface ethernet 0
ip access-group 102 in

access-list 103 permit ip
  132.5.2.0 0.0.0.255 0.0.0.0 255.255.255.255
interface ethernet 1
ip access-group 103 in
```

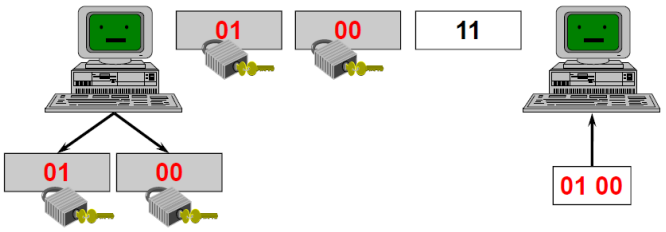
Security of network applications

The standard situation if no measures are taken is very negative, because most of systems use a very weak authentication based on username and password (password can be intercepted) or based on IP address (IP spoofing). Even if stronger systems are used (e.g., OTP, challenge response) even if these are preventing problem with authentication, there are problems with data: *data snooping/forging, shadow server/MITM, replay, filtering*.

There are two possible different approaches to avoid problems:

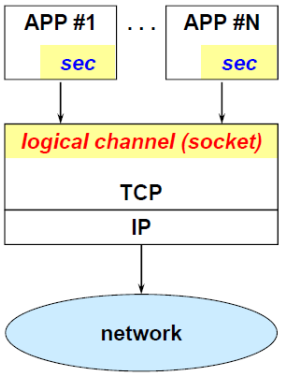


- **channel security:** it means that two nodes, before starting the communication, they negotiate algorithms, parameters, and keys to protect the whole traffic that will be sent among them through a secure channel. Since all these features are negotiated before transmitting data it is possible to get **single or mutual** authentication, **integrity,** and **privacy**. The main issue is that they are provided **only during the transit inside the communication channel**. Since when data exit the secure channel, they are no more protected and there is no possibility to achieve *nonrepudiation*. Since this is very easy to implement because it can be even implemented at OS level, it is a very much adopted solution because it does not require nearly any modification to applications. Watching the picture there are also other bits (the two 1s in black are not part of the user message) transmitted over the channel and, because of this, they get security even if they do not need security;

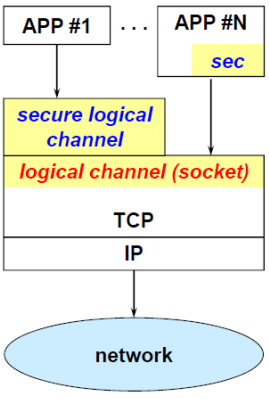


- **message/data security**: each data is individually protected by wrapping it inside secure container (e.g., PK-SSL). In this case the sender is creating protection and data that do not require security are not secured. In this case only **single** authentication is achieved, but still have **integrity and privacy self-contained in the message**. The protection remains also when the data is exiting the network and it is stored at the destination: this permits the **non-repudiation,** but it requires some modification of applications.

It is possible to mix the two approaches and have secure data inside secure channel: however, using a secure network channel only is the preferred choice, especially for networked application such as the Web.



Any network application has a common part which was basically the TCP/IP stack with the logical interface which is named **socket** (it permits to send and receive TCP or UDP data in a standard way). So that, when applications needed security, they had something inside itself to implement the security part. Each application implements **security internally** and the common part is limited to the communication channels (i.e., the socket). There could be **possible implementation errors** (inventing security protocols is not simple) and **interoperability is not guaranteed** due to possible different interpretation of specifications.



This was a solution but today is no more largely used, because for compatibility reasons it implied to buy everything from the same vendor.

The other way is to implement **security externally** to applications. The **session level** would be the ideal place to be used to implement many security functions, but session

level does not exist in TCP/IP. Then, someone proposed the **"*secure logical channel*" level**. It is something which uses socket to send data enriched with some protection. It simplifies the work of application developers; it avoids implementation errors, and it is up to the application whether to select it or not (no compatibility issues). Today it is a de facto standard.

Secure channel protocols

- **SSL/TLS**
  - The most widely used: initially was SSL, then renamed TLS.
- **SSH**
  - It was a successful product (especially in the period when export of USA crypto products was restricted), but today it is a niche product to be used only in certain restricted environments.
- **PCT**
  - Proposed by Microsoft as an alternative to SSL. It is one of the few fiascos of Microsoft because never became famous and MS dropped PCT from its products.

SSL (Secure Socket Layer)

Initially named **Secure Socket Layer**, SSL was proposed by *Netscape Communications* (inventor of the first graphical browser for computers) which needed a secure protocol to implement e-commerce on the web. SSL is a **secure transport channel** (session level) with some properties:

- **Peer authentication (server, server and client)**
  - It is compulsory to have server authentication when the SSL channel is opened, due to its origin: it was needed for e-commerce. Optionally it is possible to also request client authentication. Remember it is at session level, which means that username/password is not required.
- **Message confidentiality**
  - Confidentiality is optional because maybe authentication is enough.
- **Message authentication and integrity**
  - Authentication and integrity are compulsory because are used to demonstrate who created the message and that the message has not been modified during the transmission.
- **Protection against replay and filtering attacks**
  - If someone is taking a copy of the message and sending again it to the channel, it will be detected as a replay attack. In the same, if an intermediate node is deleting (filtering) part of the data, the receiver will detect that there is a missing part.

Since these features are laid on top of TCP, the success of SSL is due to the fact it is easily applicable to all protocols based on TCP: *HTTP*, *SMTP*, *NNTP*, *FTP*, *TELNET*. An example is the famous *HTTPS* on *443/TCP*.

```
nsiiops       261/tcp # IIOP Name Service over TLS/SSL
https         443/tcp # http protocol over TLS/SSL
smtps         465/tcp # smtp protocol over TLS/SSL (was ssmtp)
nntps         563/tcp # nntp protocol over TLS/SSL (was snntp)
imap4-ssl     585/tcp # IMAP4+SSL (use 993 instead)
sshell        614/tcp # SSLshell
ldaps         636/tcp # ldap protocol over TLS/SSL (was sldap)
ftps-data     989/tcp # ftp protocol, data, over TLS/SSL
ftps          990/tcp # ftp protocol, control, over TLS/SSL
telnets       992/tcp # telnet protocol over TLS/SSL
imaps         993/tcp # imap4 protocol over TLS/SSL
ircs          994/tcp # irc protocol over TLS/SSL
pop3s         995/tcp # pop3 protocol over TLS/SSL (was spop3)
msft-gc-ssl   3269/tcp # MS Global Catalog with LDAP/SSL
```

Figure 61 Official ports for SSL applications

SSL – authentication and integrity

Peer authentication at channel setup is performed with strong authentication:

- The **server authenticates itself** by **sending its public key** (*X.509 certificate*) AND by **responding to an implicit asymmetric challenge**
- The client authentication (with public key, X.509 certificate and explicit challenge to make client use the private key in order to demonstrate the possession of it) is optional (since a lot of people don't own a X.509 certificate)

If peer authentication fails, the channel is not opened. If it is successful, then all data are protected.

For **authentication and integrity** of the data exchanged over the channel, the protocol uses:

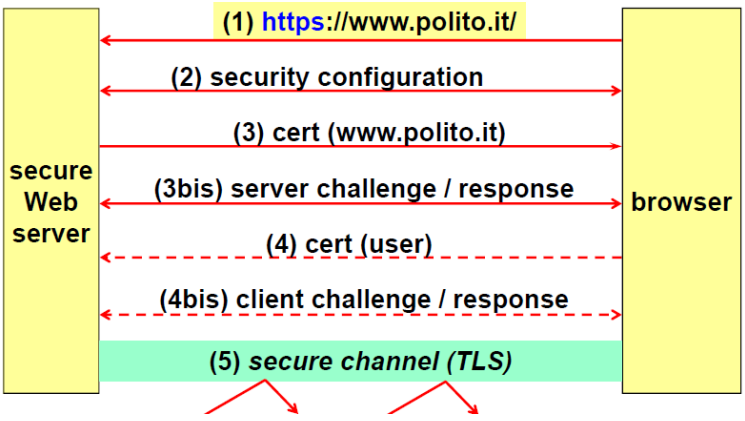
- *A **keyed digest** (SHA-1 or better)* to demonstrate authentication and integrity*;*
- *A **Message IDentifier** i.e., a number to avoid reply and cancellation:* it is possible to have because SSL is layered on TCP and there are never missing, duplicate or out of order messages within TCP. For example, an application message with the same identifier of another one is surely the result of a reply attack, because in case of missing packet TCP would have taken care of that before delivering the data at the upper level. Similarly, if after receiving message with `id=1` the application receives a message with `id=3`, then surely this is not due to a missing packet, but it is a cancellation attack, in fact TCP would have taken care of retransmitting the lost (or delayed) packet.

SSL – confidentiality

Optionally is possible to have confidentiality:

- *The client generates a session key later used for symmetric encryption of data* (RC4, 3DES, IDEA, AES, …);
- *Key **exchange** with the server occurs via **public key cryptography*** (RSA, Diffie-Hellman or Fortezza-KEA, a variant of Diffie-Hellman used in the past by US military personnel).

TLS



In the picture, there are a *secure web server* and a *browser*. The browser wants to open a secure channel by using HTTPS. The first step is to **agree about a security configuration**. Since there are several algorithms that can be used, the browser and the server will declare their list of algorithms and they must find a match. At this step, connection *could* fail (if a common language is not found). **TLS is a negotiation protocol**. Next, if they have agreed about a set of algorithms, the server would give back its certificate which must contain the name corresponding to the connected URL. Then, the server must use its private key to implicitly respond a challenge (on the picture there is a simplification, the browser is not sending a challenge). As will be clearer later, the implicit challenge consists in the decryption of a key called "pre-master secret", decided by the client and sent to the server encrypted with the public key present in the certificate.

Optionally, the server may ask browser to send its own X.509 certificate and then the server will explicitly send a challenge to the browser which will need to use the private key and provide a response. If everything is successful, the secure channel will be created.

SSL–3 architecture point of view

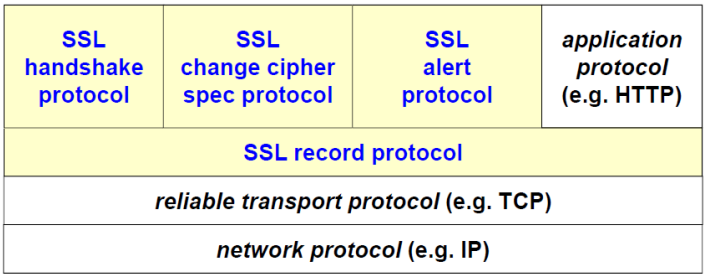
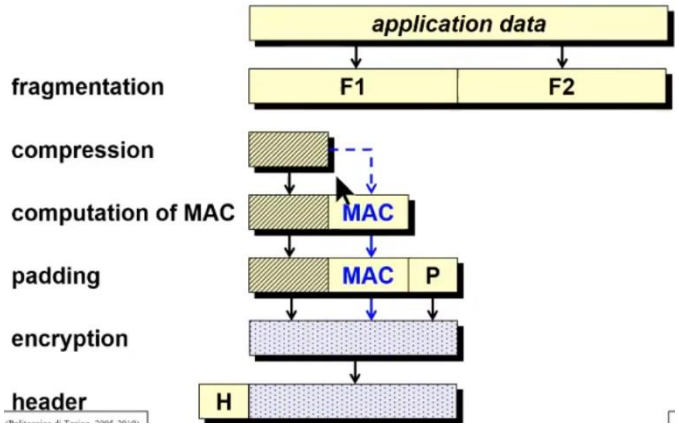


Figure 62 Partial protocol stack when using SSL.

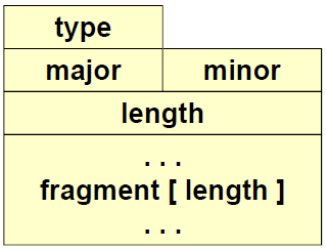
At the base there are the network protocol (e.g., IP) and a reliable transport protocol (e.g., TCP). On top of TCP there is a **record protocol** to transmit SSL records (i.e., protected data). Inside it there are application data, but initially it is used for the **handshake protocol** (i.e., the agreement phase in the previous picture). Then, there is also the **change cipher spec protocol** used at the end of the handshake to tell systems that the agreement ended. Finally, there is the **alert protocol** which is a way to send alarms (missing or duplicated packets) or just to signal the end of communication.

When the record protocol is used to send application data, there are some huge files (e.g., big file from the server) to be sent and here SSL has a limit: it is possible to send only packets of a specific length (32 Kbytes). The application data is first **fragmented** and then **compressed** (this on older versions only, because it originated attacks) to save space during transmission. Then a MAC is computed to protect authentication and integrity. If confidentiality is wanted too, there is also some padding added. As one of the last steps, the new `compressed+MAC+P` packet is encrypted and then a header is prepended to make it recognizable as an SSL segment inside TCP.



TLS-1.0 record format

- uint8 *type* = *change_cipher_spec* (20), *alert* (21), *handshake* (22), *application_data* (23)
- uint16 *version* = *major* (uint8) + *minor* (uint8)
- uint16 *length*:
  - $\leq 2^{14}$ (record not compressed) for compatibility with SSL-2
  - $\leq 2^{14} + 1024$ (compressed records)
- This means that it is a 5-byte header
- Plus a payload of max 16kB



SSL – computation of MAC

```
MAC = message_digest ( key, seq_number || type ||
version || length || fragment )
```

In order to protect authentication and integrity a MAC is computed with a specific cryptographic hash (digest algorithm) and the key that has been negotiated during the handshake. Everything is protected by this MAC: sequence number, type of packet, version, length, and the data itself (the fragment).

Some notes:

- **Message_digest**
  - It depends on the chosen algorithm;
- **Key**
  - It is different according to the direction: one key to protect data from the client to the server; one key to protect data from the server to the client. It is important because otherwise an

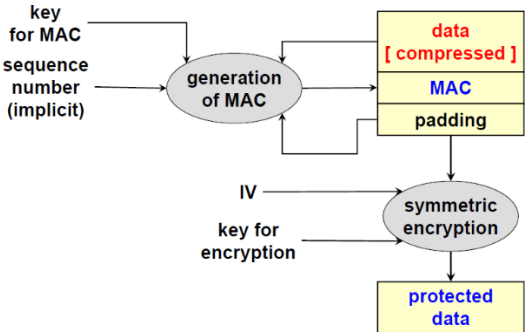
attacker could copy a packet being sent from client to server and reply that as part of the server to client.

- **Seq_number**
  - It is a 64-bit integer that **is never transmitted** but computed implicitly (because TCP is used and therefore no packets can be lost). An integer that big allows keeping the channel opened sending quite a lot of data before reaching the maximum number expressed with 64 bits. When this limit is reached, the channel must be closed and another one will be opened.

SSL/TLS handshake protocol

The handshake protocol is needed to:

- **Agree on a set of algorithms** for confidentiality and integrity;
- **Exchange random numbers** between the client and the server to be **used for** the subsequent **generation of the keys;**
- **Establish a symmetric key** by means of public key operations (RSA, DH, or Fortezza);
- **Negotiate the session-id:** used to avoid renegotiation of security parameters when contacting the same server multiple times;
- **Exchange the necessary certificates.**



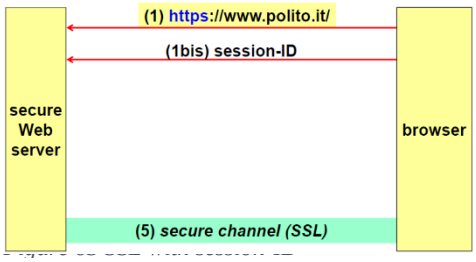
From data protection point of view there are data (optionally compressed) which need to be protected. A MAC is computed by taking data, the key, the implicit sequence number and the eventually padding. Then the MAC is inserted. The packet is then encrypted symmetrically with one IV (thinking about CBC) and a specific key for encryption. At the end, the data is protected, and SSL header will be prepended to it.

Session-id

Typical web transaction looks like the following:

- *1. Open, 2. GET page.htm, 3. page.htm, 4. Close*
- *1. Open, 2. GET home.gif, 3. home.gif, 4. Close*
- *1. Open, 2. GET logo.gif, 3. logo.gif, 4. Close*
- *1. Open, 2. GET back.jpg, 3. back.jpg, 4. Close*
- *1. Open, 2. GET music.mid, 3. music.mid, 4. Close*

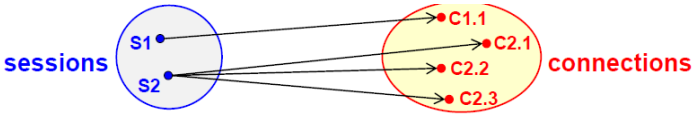
Channels are continuously open and closed! This is particularly stressful when SSL is used to open the channels because if the SSL cryptographic parameters must be negotiated every time, then the computational load (public key operations) becomes very high. To avoid re-negotiation of all the cryptographic parameters for each SSL connection, the SSL server can send a permanent identifier named **session identifier** (more connections can be part of the same logical session): this is a way to represent a set of algorithms and keys that have been negotiated at some time. If the client, when opens the SSL connection, before starting handshake, sends a valid **session-id** (already negotiated before) then the negotiation part is skipped, and data are immediately exchanged over the secure channel. Note that the server can reject the use of session-id (always or after a time passed after its issuance).



In the figure, before starting the negotiation, immediately a session-ID is sent. If the session-ID is still in server's cache, the secure channel is opened. If an attacker tries to use the same valid session-ID, it will not be able to send messages through the channel because it does not know which algorithms/keys were used and it cannot compute the correct MAC. However, the time to keep the session-

ID with all the related information should be carefully chosen to avoid server being overloaded by keeping track of user sessions. So, there are two concepts to keep in mind when dealing with SSL:

- **TLS session**
  - *It is **a logical association between client and server***
  - *Created by the Handshake Protocol*
  - *Defines a set of cryptographic parameters (algorithms, keys and so on…)*
  - *Is shared by one or more SSL connections (1: N)*
- **TLS connection**
  - *A transient TLS channel between client and server*
  - *Associated to one specific TLS session (1:1)*



Relationship among keys and sessions (between a server and the same client)

When the handshake protocol is running, two public keys are established, which are called **pre-master secret** (long-term secret) and **master secret**. When the connection is the first in the session, the pre-master secret is generated. Then, for each connection, including the first one, the client and server generate a **random number** which are combined with the pre-master secret to create the **master secret** (some kind of symmetric operation, like key derivation function). For each connection, the keys for MAC, the keys for encryption and the IV for encryption are derived by combining the master secret with the random values that are different for each connection and are exchange in clear. The only persistent thing in the same session is the master secret: the first time it will be generated starting from the pre-master secret and the random numbers that are generated every time there is a new connection.

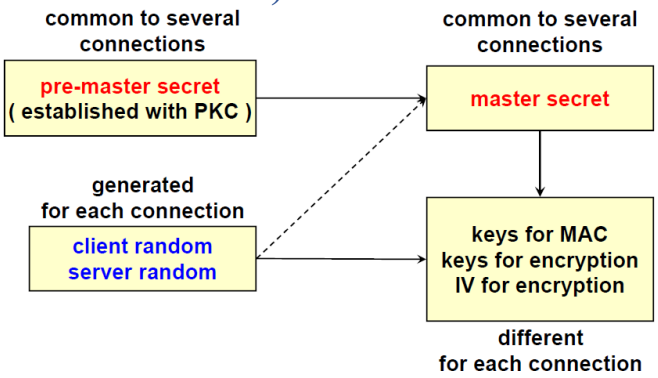


Figure 64 This figure refers to keys and secrets generation, without the mechanisms added to obtain the perfect forward secrecy.

Perfect forward secrecy

If a TLS server has a valid certificate for both signature and encryption, then it can be used both for authentication (via a signature) and key exchange (asymmetric encryption of the session key). But if an attacker copies all the encrypted traffic and later discovers the long-term private key, then the attacker can decrypt all the traffic: past, present, and future. To avoid that scenario the "perfect forward secrecy" is performed: in this scenario the compromise of a private key is compromising only the current (and eventually the future one) traffic but not the past one.

This implies that the key used for authentication should be different from the key used for encryption, but it is not enough. A key used for encryption should be a one-time key.

"Ephemeral" mechanisms

Nowadays, in order to implement PFS, for encryption there is a one-time key generated on the fly:

- **For authenticity, the key must be signed** (but cannot have an associated X.509 certificate because the CA **process is slow** and often not on-line)
- DH suitable, RSA slow (because it involves prime numbers)
  - The compromise to use an ephemeral RSA key is to generate it once and then re-use it N times (typically 10-100 times)

By generating an encryption key, which is generated on-the-fly, and signing it with the private key, there is the benefit that the server's private key is used only for signing, so we obtain perfect forward secrecy:

- If the (temporary or short-lived) private key is compromised, then the attacker can decrypt only the related traffic;
- Compromise of the long-term private key is an issue **for authentication but not for confidentiality.**

Examples: TLS servers nowadays use *ECDHE* (*Elliptic Curve Diffie Hellmann Ephemeral*) that provides that the Diffie Hellmann parameters are not fixed but generated on-the-fly.

TLS and virtual servers: the problem

TLS is a secure channel activated before the application level. This creates issues especially with virtual server (frequent case with web hosting). Given the shortage of IP addresses it is normal practice to create a virtual server which is a server with a different name but the same address of another one: e.g., *home.myweb.it=1.2.3.4*, *food.myweb.it=1.2.3.4.*

It is easily done in HTTP/1.1 because the client opens a TCP channel towards that address, and then the client sends the Host header to specify the name of the server it wants to connect to.

But it is difficult to apply in HTTPS because TLS is activated before HTTP. When browser opens TCP port 443 towards the address *1.2.3.4* which certificate should the server provide? In fact, the name in the certificate must match the name typed as URL.

TLS and virtual servers: solutions

A solution is to create **collective** (*wildcard*) **certificate** such as *CN=*.myweb.it*. The problems are that private key must be shared between all servers that share that certificate and that not all the browsers treat these kinds of special certificates in the same way (i.e., the actual effectiveness is up to the browser of the user).

In addition to the common name, it is also possible to insert a "**subject alternative name**" (*subjectAltName*) which is a list of virtual servers sharing the same IP address. This still implies that the private key must be **shared** by all servers and it need to **re-issue the certificate** at any addition or cancellation of a server.

The best solution would be to use the **SNI** (*Server Name Indication*) **extension** in ClientHello (first message sent from client to server) standardized by RFC-4366. This extension permits to the client to indicate that it will then connect to a specific server name. Unfortunately, since it is an extension, it has limited support by browsers and servers.

ALPN extension (Application-Layer Protocol Negotiation)

Another issue is related to the fact that once the TLS channel is opened, there is an application protocol which runs protected inside it. But also, the application protocols may have various versions and it is a problem. This problem is solved by the **ALPN extension** (RFC-7301).

ALPN is an Application Protocol Negotiation (for TLS-then-proto) to speed up the connection creation, avoiding additional roundtrips for application negotiation:

- *ClientHello* asserts *ALPN=true+list of supported app. protocols*
- *ServerHello* asserts *ALPN=true+selected app. protocol*

When the clients select a protocol version to use, the serve may not support that protocol: in this case, the connection is closed, and client must try again with another version (maybe many times). ALPN is a way to avoid opening many TLS channels and agree which application protocol version will be used.

This is particularly important to negotiate HTTP/2 and QUIC (which is an UDP-based version of HTTP) because Chrome and Firefox support HTTP/2 **only over TLS**. It is useful also for those servers that use different certificates for the different application protocols.

Some possible values are http/1.0, http/1.1, h2 (it stands for http/2), h2c.

DTLS

Given the success of TLS, there has been an effort to use the same idea to also protect datagrams (because TLS can be used only on level 4 reliable protocols e.g., TCP). **DTLS** is **Datagram Transport Layer Security** (RFC-4347) and it applies the TLS concepts to datagram security (e.g., UDP), but it does not offer the same properties as TLS (e.g., data cancellation, data replay because in level 3 packets can be lost or duplicated).

DTLS is in competition with IPsec and application security, so there are a few choices to make when implementing security. For example, let us talk about **SIP** (protocol used for VoIP) security. It is possible to implement security on SIP:

- With IPsec (since normally SIP uses UDP packets which are transported on IP)
- With TLS (only for SIP_over_TCP)
- With DTLS (only for SIP_over_UDP)
- With secure SIP (performs protection directly at application level)

Quite often the decision is practical: which one is easier to implement?

The TLS downgrade problem

There are various versions of TLS: the original SSL, version 2, 3 and then TLS 1.0, 1.1, 1.2, … Typically the older versions have security problems and should not be used unless explicitly required.

Normally, client sends within ClientHello message the highest supported version and server notifies (in ServerHello) the version to be used, which is the highest in common with client.

A normal version negotiation between *Client* and *Server* is like this:

- **Agreement on TLS-1.2**
  - $(C \rightarrow S)$ 3,3 (it means major version 3, minor version 3 which by default is TLS-1.2)
  - $(S \rightarrow C)$ 3,3 (server agrees with the version proposed by the client)
- **Fallback to TLS-1.1 (e.g., no TLS-1.2 at server)**
  - $(C \rightarrow S)$ 3,3
  - $(S \rightarrow C)$ 3,2 (server downgrades to a previous version)

Here there is a problem: some servers, rather than sending the correct response, close the connection; then the client has no choice but to try again with a lower protocol version. This raises to the **Downgrade attack**: the attacker sends *fake server response*, to force repeated downgrade until reaching a vulnerable version (e.g., SSL-3) and then execute a suitable attack (e.g., Poodle). But not only attacks are possible, because there could be for example an error and the connection with the server is closed due to a network problem. For that reason, a new extension has been added.

TLS fallback **S**ignalling **C**ipher **S**uite **V**alue (SCSV)

This extension is reported in RFC-7507 to **prevent** protocol downgrade attacks. This is done through the creation of a new (dummy) cipher suite which is not listing any kind of algorithms, but just mentioning *TLS_FALLBACK_SCSV* that **SHOULD** be sent by the client when opening a downgraded connection by putting it as last in cipher suite list.

Example: suppose that the client proposed 1.1 but there has been a downgrade to 1.0. The response will be sent with 1.0 but since it is possible to do better it is possible to signal that by putting FALLBACK_SCSV in the cipher suite.

If it is possible to use a better version, a new fatal alert value "*inappropriate_fallback*" is sent from the server. It **must** be sent by the server when receiving TLS_FALLBACK_SCSV and a version lower than the highest one supported. If there was an error, then the channel is closed, and the client should retry with its highest protocol version.

Many servers do not yet support SCSV, but most servers have fixed their bad behaviour when the client requests a version higher than the supported one, so browsers can now disable insecure downgrade: Firefox from 2015 and Chrome from 2016.

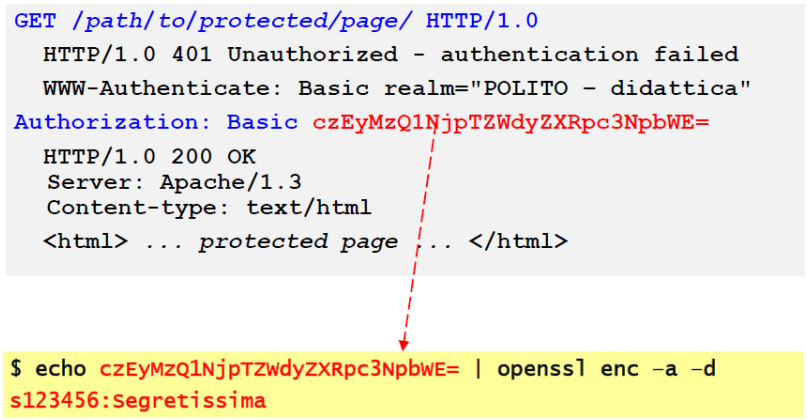
HTTP security

On top of TLS there is often HTTP. In HTTP/1.0 there are two security mechanisms defined:

- "**address-based**": the server performs **access control** based on the IP address of the client (useless because of IP spoofing);
- "**password-based**" (or *Basic Authentication Scheme*): access control based on username and password only Base64 encoded (nearly in clear, because encoding is not encryption and no key must be provided)

Both schemas are highly insecure (since HTTP assumes they are used on a secure channel!). In HTTP/1.1 has been introduced "**digest authentication**" based on a symmetric challenge response protocol, which is documented in RFC-2617 *"HTTP authentication: basic and digest access authentication"*.

HTTP – Basic Authentication



Client is sending to the server a request (e.g., GET of a page) with HTTP/1.0. A protected page looks the same as an unprotected one. The client gets an error *Unauthorized – authentication failed*. The error 401 is the only error in which the server is permitted not to close the channel, but to keep it open. The reason is because the client did not know that the page required authentication. The server continues the dialogue with a new header "*WWW-Authenticate*" which is telling the client that to get the page there must be an authentication (with basic mechanism) on the **realm** "POLITO – didattica" that in this case is a suggestion that will be displayed in a pop-up message to suggest to the client which username and password should be used.

When the browser receives this message, it opens a pop-up to provide username/password to open the page. After user inserts credentials, the browser will send the new command "*Authorization: Basic czEyMzQ1NjpTZWdyZXRpc3NpbWE=*" which is the Base64 encoded username and password. The server will compare that string with username/password stored in server's memory and if it is valid then the server will provide the originally requested page.

Since it is only encoded, it is possible to process it with openssl without requiring any key. Anybody could get credentials (unless inside an encrypted channel).

HTTP digest authentication

HTTP 1.1 added a new way of authenticating client, which is reported in RFC-2069 (technically obsoleted, but it is still the base case in RFC-2617). The challenge is implicit, and the response must use the password.

The digest computation is the following:

- The browser computes a digest: *HA1 = md5(A1) = md5(user ":" realm ":" pwd);*
- Then the browser computes another digest: *HA2 = md5(A2) = md5(access method (e.g., GET) ":" URL);*
- Lastly the browser sends to server: *response = md5(HA1 ":" nonce ":" HA2)*
    - The nonce is generated by the server and it is sent in the 401 response: its purpose is to avoid replay attacks;

- The authentication server may insert a field "opaque" to **transport state information** (e.g., a SAML token) towards the content server.

```
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"
  HTTP/1.1 200 OK
  Server: NCSA/1.3
  Content-type: text/html
  <HTML> pagina protetta ... </HTML>
```

pwd=antonio

In this case, the GET is made with protocol HTTP/1.1. The server has two options: authentication can be made with the base or with the digest. In this case, the server gives back the *401 error* and then proposes the digest authentication by providing *the realm, nonce and opaque*.

The browser will ask for username and password and then send back the blue string reported in the picture. The server will then answer back with the requested page.

HTTP and SSL/TLS

There are two different approaches for combining HTTP with SSL/TLS:

1. *"TLS then HTTP" (RFC-2818 – HTTP over TLS)*: a TCP channel is opened, on top of it TLS is created and when TLS is active then HTTP starts;
2. *"HTTP then TLS" (RFC-2817 – upgrading to TLS within HTTP/1.1)*: two TCP ports 80 can be connected and start talking HTTP, when there is some data that requires protection a normal channel can be transformed in a secure one. This is done with a special command (`start TLS`) which would permit browsers and servers to transform the channel from a plain one to a TLS.
3. *Note: "SSL then HTTP" is in widespread use but it is undocumented*

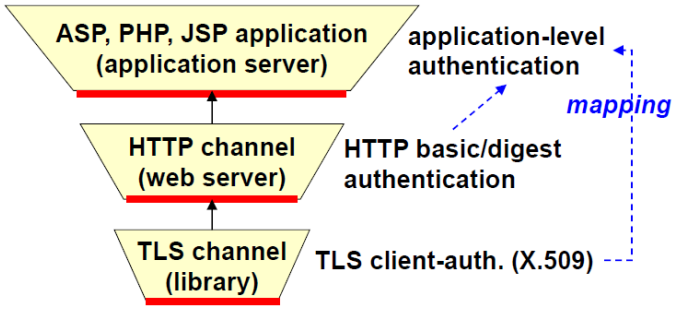
The two main approaches are not equivalent in terms of final result (i.e., it will be not the same) but of course both of them converge to a secure channel, plus they have an impact over: applications, firewall and IDS (*Intrusion Detection System*).

- **Applications**: if you have TLS then HTTP, the application developer has no issues because TLS is managed by the system manager of the node or by the network manager and the web developer knows that it is protected and has no problem about that. In this case TLS is always up, even for pages that does not require TLS.
  On the contrary, with HTTP then TLS, the web developer has the option to activate TLS when needed, but then the application developer is in charge of this part and he must understand what he is doing and manage it correctly.
- **Firewall**: it is an element which is filtering traffic (it decides what can pass through and what must be stopped). If TLS then HTTP is used, it means that there are two different ports: port TCP 80 for HTTP, port TCP 443 for HTTP over TLS. The firewall can easily distinguish the traffic: traffic over port 80 is not protected, traffic over port 443 is protected.
  On the contrary, if there is HTTP then TLS, firewall cannot make any check because only port 80 for HTTP is used and then later the developer can decide to turn that from plain to protected channel. Firewall is no more in charge of selecting only secure channels.
- **IDS**: it is a system which inspects the traffic. In case of IDS if TLS then HTTP is used, the IDS is not able to inspect anything, because the traffic is encrypted.
  On the contrary, if HTTP then TLS is used, IDS can at least have a look at the initial part and see when the channel is turned to a secure one.

TLS client authentication at the application level

Via client authentication it is possible to identify the user that opened the TLS channel (without asking for his username and password). Some web servers support a (semi-)automatic mapping between the credentials extracted from the X.509 certificate and the users of the HTTP service and/or the OS. When authentication is performed at that level, it is automatically recognized also at application level.

Authentication in web applications



When there is a web application layered on TLS, there are several levels, reported in the picture, in which authentication can be performed. The important point is that the earlier is the access control, the smaller is the attack surface.

If TLS is enabled, the first part used for connection is the **TLS library** (which is quite small, it just implements TLS) and the small red part is the one that can be attacked.

If TLS client authentication is enabled, it means that if the attacker does not have an X.509 certificate, or it is not authorized for access, the attack will be rejected at that level and the only part of the system that can be attacked is the TLS library itself. If TLS is activated but there is no client authentication, then the TLS library can be attacked, but also the web server itself (because before going to the web application, HTTP is needed but an HTTP server is a quite big piece of software, which has more bugs than a TLS library). It is possible to perform authentication in this level with *basic/digest authentication* and if the attacker fails at this level it will not proceed (and channel will be closed). If HTTP is implemented without authentication, then the application server is reached, which is the page that the developer has written (ASP, PHP, JSP) with forms (username/password requested to user). At this level is too late, because the attacker can try to use bugs in TLS library, HTTP server and bugs in the pages of the application. For that reason, authentication should be performed as soon as possible to reduce the attack surface. Beware that there is always the possibility to perform a mapping from the TLS authentication or from the HTTP authentication and bring that to the application pages (*example from Prof. Lioy performed on PoliTo's website in Lesson 16 on 20th Nov 2020*).

What about forms requesting user/password?

Technically speaking, it is not important if the page containing the form is secure (e.g., *http://www.ecomm.it/login.html*) because the actual security depends on the URL of the method used to send username and password to the server (e.g., *<form … action=https://www.ecomm.it/login.php>*).

If the page is not secured, user could be exposed to **phishing**, because other people could create a fake web page with similar URL address, but with another action performed in the form and only few uses have the technical knowledge to verify the URL of the HTTP method used to send user/password. There is no proof that we are on the correct website if HTTPS is not used.

HTTP Strict Transport Security (HSTS)

In HTTP a new header has been inserted, which is reported in RFC-6797 and it is named **HSTS**. Via this header, the HTTP server declares that all its interaction with UA (user agent) must only be done via HTTPS.

Features:

- Prevents protocol downgrade and cookie hijacking
- It is valid only in HTTPS response
- The header has an expiration that is renewed at every access
- May include subdomains (recommended)
- May be pre-loaded (i.e., browsers may have already knowledge that when a certain domain is contacted TLS must be used):
    - It is a bit dangerous because if HTTPS fails there is no possibility to contact server

- Preload list maintained by Google and used by many browsers = https://hstspreload.org/

HSTS – syntax and examples

```
Strict-Transport-Security:
   max-age = <expire-time-in-seconds>
   [ ; includeSubDomains ]
   [ ; preload ]
```

```
$ curl -s -D- https://www.paypal.com/ | fgrep -i strict
strict-transport-security: max-age=63072000
$ curl -s -D- https://accounts.google.com/ | grep -i
strict
strict-transport-security: max-age=31536000;
includeSubDomains
```

The HTTP server as a response can insert the syntax in the picture ("*Strict-Transport-Security*") which tells the expire time and optionally includes subdomains and pre-load actions.

In the second part of the picture, there is a check if anybody is using HSTS. The first example is for PayPal and only the "strict" string is grep. The answer is the one reported in the picture, which means that for a long time PayPal will use only TLS. The second example is performed on Google. The answer is the same, but it also includes subdomains.

HTTP Public Key Pinning (HPKP)

To avoid someone to create a fake certificate for a website, the manager of the website can insert in the header the hash of the public key that it is using, and the UA (browser) caches this key and will refuse to connect to a site with a different key, even if that website is presenting a valid certificate. This is needed because in the past the certification authority was tricked to create a certificate for another server (RFC-7469).

This is a TOFU (*Trust On First Use*) technique which is dangerous when losing control of the key or when the key must be updated (always include at least one backup/secondary key). The header permits also to specify a URI where the browsers can report violations. The header can be used in **enforcing** or **report-only** mode.

- **Enforcing**: if a server does not have the public key, the connection is closed.
- **Report-only**: if the server has a different public key, the connection proceeds but it is reported to user.

The syntax is reported on the right. There is the computation of base64-sha256 of public-key that is wanted to be fix and it is possible to specify the expire time, subdomains, and the report-URI.

```
Public-Key-Pins:
   pin-sha256 = " <base64-sha256-of-public-key> ";
   max-age = <expireTime-in-seconds>
   [; includeSubDomains]
   [; report-uri = " <reportURI> "]
```

To use Report-only and not Enforcing, the second header in the picture must be used. The syntax is the same.

```
Public-Key-Pins-Report-Only:
   pin-sha256 = " <base64-sha256-of-public-key> ";
   max-age = <expireTime-in-seconds>
   [; includeSubDomains]
   [; report-uri = " <reportURI> "]
```

```
$ $ curl -s -D - https://scotthelme.co.uk/
    | fgrep -i public-key
public-key-pins:
pin-sha256="9dNiZZueNZmyaf3pTkXxDgOzLkjKvI+Nza0ACF5IDwg=";
pin-sha256="X3pGTSOuJeEVw989IJ/cEtXUEmy52zs1TZQrU06KUKg=";
pin-sha256="V+J+71HvE6X0pqGKVqLtxuvk+0f+xowyr3obtq8tbSw=";
pin-sha256="91BW+k9EF6yyG9413/fPiHhQy5Ok4UI5sBpBTuOaa/U=";
pin-sha256="ipMu2Xu72A086/35thucbjLfrPaSjuw4HIjSWsxqkb8=";
pin-sha256="+5JdLySIa9rS6xJM+2KHN9CatGKln78GjnDpf4WmI3g=";
pin-sha256="MWfCxyqG2b5RBmYFQuLllhQvYZ3mjZghXTRn9BL9q10=";
includeSubDomains; max-age=2592000;
report-uri="https://scotthelme.report-uri.com/r/d/hpkp/
   enforce"
public-key-pins-report-only:
pin-sha256="X3pGTSOuJeEVw989IJ/cEtXUEmy52zs1TZQrU06KUKg=";
pin-sha256="Vjs8r4z+80wjNcr1YKepWQboSIRi63WsWXhIMN+eWys=";
pin-sha256="IQBnNBEiFuhj+8x6X8XLgh01V9Ic5/V3IRQLNFFc7v4=";
max-age=2592000;
report-uri="https://scotthelme.report-uri.com/r/d/hpkp/
   reportOnly"
```

An example is run on the left with a security blog website, which uses several public keys for backup. It includes subdomains and includes the report-URI for Enforce mode.

Then there is a second section, with only three keys and a different URI to report the problem when it is not compulsory.

E-payment systems

The motivations for dedicated payment protocols are:

- Failure of the digital cash, for technical and political problems (e.g., the DigiCash failure);
- Failure of a dedicated payment protocol (SET, Secure Electronic Transactions) due to technical and organizational problems;
- Currently the most widely used approach is transmitting a credit card number over a TLS channel but this is not a guarantee against fraud: in the past years VISA Europe declared that Internet transactions generate about 50% of the fraud attempts, although at that time they were just 2% of its total transaction amount!
    - This means that Internet is not the place where credit card numbers are stolen, but it is the place where it is easier to use a stolen credit card number;
    - The fact that TLS protects the transmission of the credit card number is not a guarantee, because the problem is related to the fact the data has been stolen before (and used in Internet) or that the server is untrusted.

A web-based payment architecture

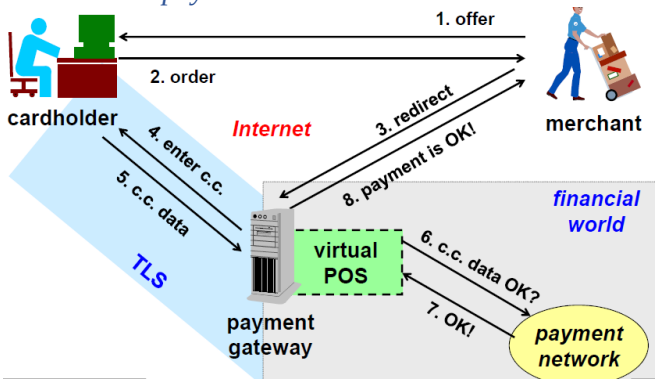


Figure 65 The most common architecture when implementing payment with credit card in Internet

The first interaction in this scheme is the offer by the merchant: just imagine the cardholder browsed the shop, filled a market basket and now he is ready to perform the check out. When the cardholder will place the order, the recommended setting is that the merchant is not asking for the data about the payment, because it is too risky and would require him to guarantee very high security standards. The best solution is that the merchant has entered into an agreement with a payment gateway and redirecting the request to it. At this point the cardholder is not in contact with the merchant anymore, but with the payment gateway, that creates a secure channel, over which the data of the credit card are requested and sent back. Here the problem is to validate these data: if that were a physical payment, a **POS** (*Point Of Sale*) device would have been used, for this reason in this case the payment gateway creates a **virtual POS** which, starting from the credit card data, creates a transaction valid for the payment network, to check if the data of the credit card are valid for that specific payment. When the payment network provides a positive answer to that check, then the gateway will provide a positive answer to the merchant, who will then be able to provide the cardholder with the required goods.

So, to summarize the baseline of this architecture is:

- the buyer owns a credit card;
- the buyer has a TLS-enabled browser.

Nevertheless, the effective security depends upon the configuration of both the server (payment gateway) and the client: TLS is a negotiation protocol, so the important thing is the **actual security level** that is negotiated between the security gateway and the user browser. Moreover, the payment gateway has all the information (payment + goods) while merchant knows only info about the goods: this is a good practice since it is good to provide the user the list of the goods that he is buying when he inserts the credit card number, but it has the drawback that the gateway could profile him, and it could not respect his privacy.

In case the merchant does not want to rely on this suggested architecture but prefers performing the payment by himself, then it should comply with the standard **PCI DSS**.

PCI DSS

PCI DSS stands for **P**ayment **C**ard **I**ndustry **D**ata **S**ecurity **S**tandard, and it is required by all credit card issuers for Internet-based transactions. It is very detailed set of technical prescriptions compared to other security standards (e.g., HIPAA = Health Insurance Portability and Accountability Act).

It is quite a complex standard, so most companies prefer not to implement it by themselves but relying on the security gateway. There are notable exceptions, such as Amazon.

There have been some versions, with some updates:

- **v2.0** = *Oct 2010*;
- **v3.0** = N*ov 2013*;
- **v3.1** = *Apr 2015* (**no SSL or "old" TLS**);
- **v3.2** = *Apr 2016* (MFA, test functions/procedures, describe architecture, ...).

The prescriptions of this standard are:

- **design, build and operate a protected network**:
  - 1st requirement: install and maintain a configuration with firewall to protect external access to the cardholders' data;
  - 2nd requirement: do not use pre-defined system passwords or other security parameters set by the manufacturer.
- **protect the cardholders' data**:
  - 3rd requirement: protect the stored cardholders' data. It is possible to encrypt the data when stored on disk or provide some strong access control, so that only allowed personnel can read the cardholder's data. The standard does not enforce a solution, the important point is being able to demonstrate to have an adequate protection;
  - 4th requirement: encrypt the cardholders' data when transmitted across an open public network. This is mandatory.
- **establish and follow a program for vulnerability management**:
  - 5th requirement: use an antivirus and regularly update it;
  - 6th requirement: develop and maintain protected applications and systems: it implies demonstrating that security has been considered from the beginning of the development and that there is a continuous verification and maintenance of security.
- **implement strong access control**:
  - 7th requirement: limit the access to the cardholders' data only to those needed for a specific task;
  - 8th requirement: assign a single unique ID to each user. This means that, within the company, each user must have one single username, no multiple authentication systems allowed. The reason behind this requirement is that if there are multiple usernames associated with the same user it becomes very difficult to assign or remove permission and trace actions.
  - 9th requirement: limit physical access to the cardholders' data.
- **regularly monitor and test the networks**:
  - 10th requirement: monitor and track all accesses to network resources and cardholders' data. It means that, as a minimum, secure log files are required to permit to know who has had access to which server and which data;
  - 11th requirement: periodically test the protection systems and procedures;
- **adopt a Security Policy**:
  - 12th requirement: adopt a Security Policy i.e., there must be a well-defined security plan and not just an attempt to implement some security features updated only after successful attacks!

Firewall and IDS/ISP

What is a firewall

A firewall is a "**wall to protect against fire propagation**": this concept comes from a regulation that is common in countries where houses are built mostly using wood, by which a wall made of brick must be interposed between two houses in some conditions to avoid the propagation of the fire. Similarly, logically speaking, in information systems security a firewall is something that is placed in a way such that, once a network is attacked (that is "on fire"), then the attack would not easily propagate to another network that is attached to it: this is the case when two networks have different security levels, so one is untrusted and an attack on it could propagate to the other more secure one.

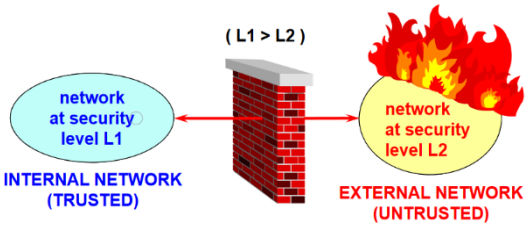


Figure 66 Typical representation of a network filter (or boundary protection)

So, besides the fact that firewall must be put at the boundary of networks with different security levels, another important aspect to consider is the directionality of the flow being controlled:

- **ingress firewall**:
  - It is a filter that checks **incoming connections** from the untrusted network;
  - The purpose is typically to select the (public) services offered; limits which internal services are offered to the external network.
  - The problem is that sometimes the request to open a connection is part of an application exchange initiated by internal users;
- **egress firewall**:
  - It is a filter that checks **outgoing connections**;
  - The purpose is typically to check the activity of the personnel: this can include control of websites visited, to verify that confidential documents are not sent outside the network or to verify that a user is not downloading some dangerous files.

This classification is easy for channel-based services (e.g., TCP applications), but difficult for message-based services (e.g., ICMP, UDP applications): in those case typically the firewall is bidirectional, in fact a distinction between ingress and egress would be meaningless.

Firewall design

When needing a firewall, one aspect to consider is that the firewall is not a single object, it is made by components, it is a system: "**you don't "buy" a firewall, you design it (you can buy its components)!"** The design can consist of a single component, and for this reason companies say that are offering firewalls: actually, they are offering single components, and typically to build a strong firewall it is necessary to compose many of them. The configuration of components is such that allows to achieve an optimal trade-off between security and functionality, with minimum cost.

Security, functionalities and the three commandments of Firewall

When designing a firewall, the first thing to consider is the security level wanted. The problem is that there is an inverse relationship with level of functionality offered: the higher the protection level, the less the offered functionalities. So, all the solutions are a compromise between these two objectives. Moreover, in designing a firewall one should always follow same principle made explicit by the invertors of the firewall (*D. Cheswick and S. Bellovin*):

- the FW must be **the only contact point** of the internal network with the external one;
- **only the "authorized" traffic can traverse the FW**: this implies having a priori the precise list of the kind of traffic, protocol or ports or even external nodes to which access should be permitted, there is the risk to cut out valid traffic (so blocking functionalities) or to be too loose and open the door for attacks;

- **the FW must be a highly secure system itself**: most companies use a powerful general-purpose computer to load into it software for the firewall, but when soon realize that the machine is most time idle, they start loading into it more components (e.g., web server, database server). This must be avoided because the more software is installed, the higher the chance to have bugs that can be exploited to perform an attack.

Authorization policies

For the second principle, in designing a firewall the authorized traffic must be identified. In expressing the authorization policy, there are two possible strategies:

- **whitelisting**: "***All that is not explicitly permitted, is forbidden***"
  - the firewall opens a few kinds of communication, hence leading to higher security;
  - more difficult to manage, especially if the users were used to have free connectivity. This could imply proving a good deal of explanation about things that are not permitted anymore.
  - It is the suggested one;
- **blacklisting**: "***All that is not explicitly forbidden, is permitted***"
  - implies studying what can be a security problem and forbidding that kind of traffic, and typically leads to lower security (open gates), because is like saying that the vulnerability is kept until it is discovered;
  - easier to manage.

Basic components of a firewall

As said, a firewall is an architecture made up by several components:

- **screening router** (*choke*): router that filters traffic at network level;
- **bastion host**: secure system, with (periodically) auditing. It is the first line of defence against attacks;
- **application gateway** (*proxy*): service that works on behalf of an application for performing communication with outside of the network, with *access control*;
- **dual-homed gateway**: system with two network cards, to act as a bridge between two different networks, but with normal routing disabled: one should put on top of it some components which are able to decide if the traffic can be safely forwarded or should be blocked.

Depending on which is the network stack layer considered by a firewall, there are different terms used to refer to it:

- **packet filter**: it means that the network level is the one used to analyse the traffic, and typically only the information contained in that level is exploited;
- **circuit gateway**: works at layer 4, so considering a TCP stream or a UDP datagram as the unit to be considered in performing filtering;
- **application gateway**: examines in detail the application data;

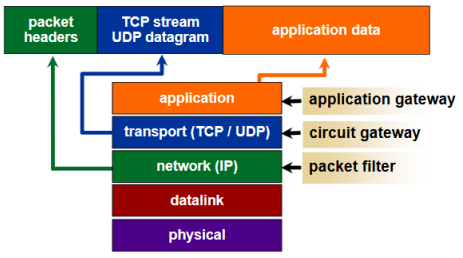


Figure 67 At which level the controls are made?

Usually, the higher we go in the stack, the more accurate is the detection of the attacks, but the slower it will be.

"Screening router" architecture

The core concept is to equip the switching element with filtering capabilities: typically, since this is a L3 network device, the kind of filtering that can perform is at IP and upper levels, hence implementing a *packet filter*. The advantage of this architecture is that there is no need for dedicated hardware; moreover, since this filter does not address in any way the

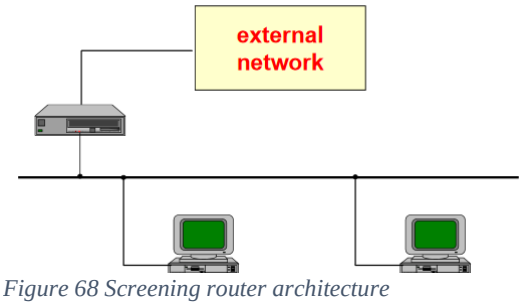


Figure 68 Screening router architecture

application level, there is no need for a proxy and hence no need to modify the applications. It is a solution simple, easy, cheap and... *insecure*, since the kind of controls it can make are very trivial, usually based on protocols, ports, and addresses. One final drawback is that the router is a single point of failure, meaning that a bug could make the attacker bypass the control and access the internal network.

"Dual-homed gateway" architecture

Because relying solely on filtering at network layer does not provide high security, to improve the architecture another element is needed to be deployed, which performs inspections at higher levels. In this architecture, the traffic permitted by the packet filter will not directly enter the internal network, but it will further be checked by a second element, generally speaking called a "gateway", which is actually a "dual-homed" because it has got two network cards, with routing disabled since a packet coming from an interface goes to the other interface if it respects the imposed rules.

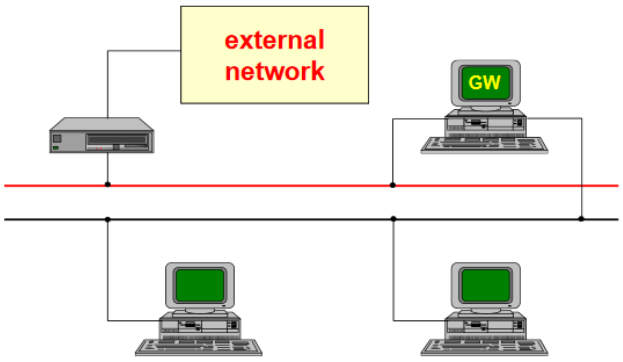


Figure 69 Dual-homed gateway architecture

This is still easy to implement, because there are two different components that are dealing with separated aspects. It requires small additional hardware requirements, just a general-purpose machine with the proper gateway software installed will suffice. Moreover, since the gateway is the frontend for the whole internal network it can masquerade the internal addresses, even without using a NAT.

The big problem is that this solution is rather inflexible, because even if a traffic should be permitted in the packet filter, then it must also pass the check at the gateway: taking as example electronic mail, typically in incoming mail it is difficult to limit who is the sender (other than forbidding well-known sites that send spam), so it is checked at the mail server, that is internal to the network. The gateway would reconstruct the application-level packet just to see that is a mail and that is not his job to inspect is, so it will send to the internal network: the inflexibility is in the fact that a packet is double checked even if the second check is useless because nothing can be decided with the information that the gateway has. The actions performed by the gateway leads to large work overhead. One thing that should be noted is that this solution implements a **double line of defence**, so it is an application of the "*defence in depth*" principle.

"Screened host" architecture

Improving the "dual-homed gateway" architecture means avoiding the bottleneck at the gateway: in the "screened host" architecture, the packet filter is also connected to the internal network but there is also the gateway, which is connected to the packet filter, so that packets that do not need a double check will be directly forwarded to the internal network. In this way there is again the problem of single point of failure in the packet filter, and a double line of defence is implemented only for those packets that need to be processed more in depth.

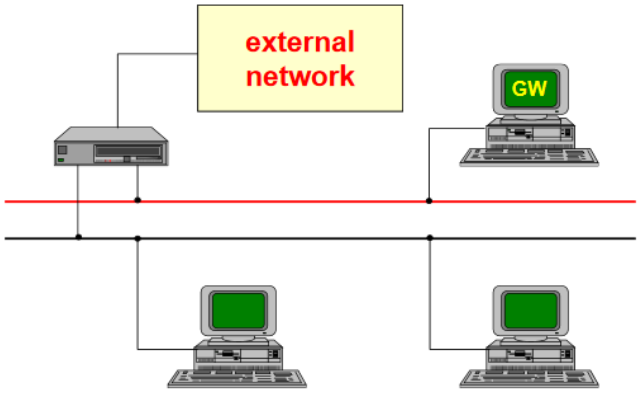


Figure 70 Screened host architecture

So, to summarize:

- the **router**:
  - blocks traffic INT ← EXT unless traffic comes from the bastion;
  - blocks traffic EXT ← INT unless traffic goes to the bastion;
  - exception: directly enabled services.
- The **bastion host** runs circuit/application gateway to control the authorized services;

- more expensive and complex to manage, because the work of the two systems must be synchronized;
- more flexible (no control over some services / hosts);
- only the hosts/protocols passing through the bastion can be masked (unless the router uses NAT).

"Screened subnet" architecture

The main problem introduced with the "screened host" architecture is the single point of failure in the packet filter. In the "screened subnet" architecture, the packet filter is split, because it performs conceptually two different operations:

- deciding if an incoming traffic should be permitted or must be denied;
- decide if traffic should be forwarded to the internal network.

In this solution, one packet filter is connected to the external network, while the other one acts like a bridge from the "intermediate" network and the internal one. First an incoming packet is checked if it is permitted, and if it is permitted directly, it is forwarded to the second packet filter that will still verify if the rule is valid and then forward to the internal network: in this way there is a double defence also for packet that will be directly transmitted to the internal network. For packets that need further inspections, there will be three defence lines, since the second packet filter will send it to the bastion. Please note that, to have a proper double line of defence implemented by the two routers, they should be sold from different vendors, because if they are equal maybe also the bugs are the same, so this will affect the security.

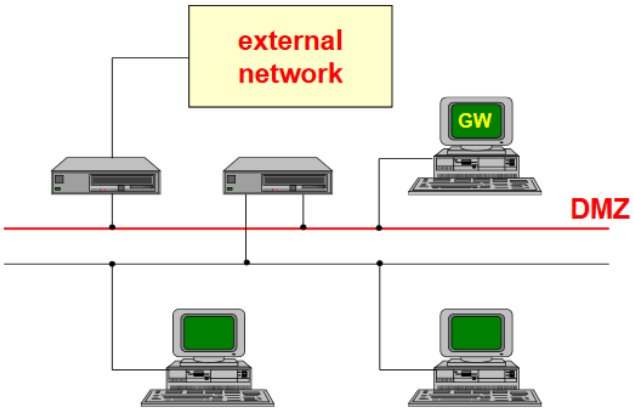


Figure 71 Screened subnet architecture

In this scheme there is a network that is completely decoupled from both the external and the internal network (the one in red in the figure): this is called **DMZ** (*De-Militarized Zone*). Typically, the DMZ is home not only to the gateway but also to other hosts (typically the public servers, such as a web server or remote access systems) and this means that public servers should be placed in DMZ. This solution is more expensive than the others, because is more complex.

"Screened subnet" architecture (version 2)

To reduce costs and simplify the management often the routers are omitted, and their function incorporated into the gateway: this solution is also called "**three-legged firewall**". In this solution the firewall is a single element, typically a general-purpose computer, equipped with three network cards: one connects to the external network, one to the internal and the third one implements the DMZ.

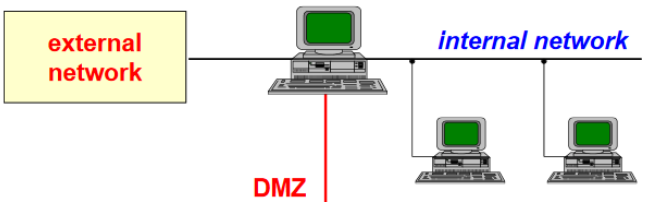


Figure 72 Screened subnet architecture (version 2)

The components of the previous solution here are software modules inside a single element, so from a functional point of view there are the same functionalities, but the solution is intrinsically less secure, because:

- *There is again a single point of failure, the single machine hosting all the processes*;
- *The complexity of this node makes room for attacks due to implementation or configuration errors*.

Despite these considerations, many companies tend to implement this solution because it gives a lot of functionalities while the cost is limited, and the management is simple because there is only an interface for managing everything. Some companies claim to have implemented "*four-legged firewall*" or "*five-legged firewall*": the number of legs comes from having multiple interfaces towards the internal network or from

deploying multiple DMZs, each one can be assigned to a different department of the company, mostly for administrative decisions.

Firewall technologies

Depending on the network level, at which controls are performed, there are different terms:

- *(static) packet filter*
- *Stateful (dynamic) packet filter*
- *Cut-off proxy*
- *Circuit-level gateway/proxy*
- *Application-level gateway/proxy*
- *Stateful inspection*

Differences are in terms of:

- *Controls to be performed (=threats detected):*
  - They are related to the threats that can be detected
- *Performance:*
  - Understanding different technologies that different level of performance can provide is important, typically because security has a price.
- *Protection of the firewall O.S.:*
  - Various solutions have different protection of the firewall itself
- *Keeping or breaking the **client-server model:***
  - The discussed firewalls are placed in between the client and the server, so it could be transparent, meaning that if the traffic is permitted the client can directly talk to the end server, or the firewall can act as a proxy, so that the client talk with the firewall and then it talks to the server. If the firewall breaks the client-server model then the solution is more secure, but the firewall itself becomes an element prone to attacks;

(Static) Packet filter

Originally it was available on routers and performed packet inspection at network level by checking just IP header and, when available, also the transport header. The reason of this is that once a packet is received it should be forwarded or discarded in the minimum possible time.

This kind of solution has pros and cons:

- **It is independent on the kind of applications**
  - *Good scalability (if throughput increases)*
  - *Approximate controls: easy to "fool" (e.g., IP spoofing, fragmented packets)*
- **Good performance**
- **Low cost (available on routers and in many OS)**
- **Difficult to support services with dynamically allocated ports (e.g., FTP)**
- **Complex to configure:** all the rules are expressed in terms of IPs, port and protocols;
- **Difficult to perform user authentication:** it comes from the fact that it uses only information available at L3;

Stateful (dynamic) packet filter

- **Conceptually similar to packet filter but "state-aware":** it means that *it remembers the previous packets in order to be faster in processing the new ones*. It can look inside packets for commands that define the ports that will be used (e.g., FTP PORT command).
  It can distinguish new connections from those already open:
  - Keeps state tables for open connections;
  - Packets matching one row in the table are forwarded without any further control (fast transmission);

- **Better performance than packet filter:**
  - SMP (*Symmetrical Multi-Processing*) support can be enabled, thanks to the use of state tables;
- **Still has many of the static packet filter limitations;**

Application-level gateway

It is composed by a set of proxies which inspect the packet payload at application level. The gateway often **requires modifications to the client application** and may optionally **mask/renumber the internal IP addresses.** When it is used as part of a firewall, usually also performs **peer authentication** (typically for egress firewall). Since the proxy is understanding the application-level commands, it can provide **top security**, for example against buffer overflow of the target application in an attack. There are differences between *forward-proxy* (egress) and *reverse-proxy* (ingress).

In general, when working at application-level there may be rules more **fine-grained and simpler** than those of a packet filter, because it is possible to express rules in terms of application-level commands and data.

Every application needs a specific proxy because it will have its own commands and data, which means:

- *Delay in supporting new applications;*
- *Heavy on computational resources*: for each connection a separate process is needed, which typically will run in user mode;
- *Low performance (because they are user-mode processes);*

It is possible to use SMP that may improve performance. It completely **breaks the client/server model** which means:

- *More protection for the server*
- *May authenticate the client*
- *Not transparent to the client:* the application needs to be configured to use the proxy;

Since it is not transparent, it means that the OS of the firewall may be exposed to attacks, because it will need to process all the packets. Additionally, there is also the problem with application-level security techniques (e.g., SSL) that will not allow to inspect the traffic. For this reason, there are some variants:

- **Transparent proxy**
  - Less intrusive for the client
  - More work (packet rerouting + destination extraction)
- **Strong application proxy** (*checking semantics, not just syntax*)
  - Only some commands/data are forwarded, for example in case of HTTP protocol, this could allow only GET and POST commands to be allowed;
  - This is the only correct configuration for a proxy (according to Lioy)

Circuit-level gateway

Between the packet filter and the application-level gateway there is also the **circuit-level**. It is a **generic proxy** (which is not "application-aware") that **creates a transport-level circuit** between client and server, but it does not understand or manipulate in any way the payload data. It just copies between its two interfaces the TCP segments or UDP datagrams (if they match the access control rules) but, in doing this, it will re-assemble the IP packets and hence it will provide protection against some L3/L4 attacks. For the server, all the attacks that are related to the TCP handshake are no more possible because there is no TCP handshake between client and server. The client will perform TCP handshake with the gateway and then the gateway will perform a correct handshake with server.

In conclusion, it breaks the TCP/UDP-level client/server model during the connection and it provides:

- **More protection for the server**:

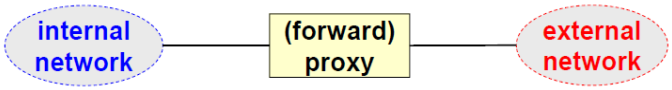
- Isolated from all attacks related to the TCP handshake;
- Isolated from all attacks related to the IP fragmentation: an attacker could fragment a packet with which performs the attack, and such an attack will not be detected by a simple packet filter;

- **May authenticate the client:**
  - But this requires modification to the application;

It still exhibits many limitations of the packet filter. The famous one is **SOCKS**.

HTTP (forward) proxy

Forward proxy, typically HTTP, is a HTTP server acting just as a front-end and then passing requests to the real server, which is external. The benefits, besides network **ACL** (**A**ccess **C**ontrol **L**ist):



- *Shared cache of external pages for all internal users;*
- *Authentication and authorization of internal users;*
- *Various controls (e.g., allowed sites, transfer direction, data types, ...);*

It is typical part of the **egress** firewall. On the contrary, the **reverse proxy** is for ingress firewall.

HTTP reverse proxy

It is an HTTP server acting just as a front-end for the real server(s) which the requests are passed to. It can again implement ACL in case it is possible to limit clients, but typically an ingress firewall does not limit clients that can contact the server. It is possible to perform **content inspection**. Benefits of this proxy are:

- **Obfuscation** (*no info about the real server*)
  - Since the server proxy is responding to the client, it can declare that is a generic proxy without providing any information to the real software that implement the final server
- **SSL accelerator** (*with unprotected back-end connections …*)
  - Reverse proxy can be the end point for SSL/TLS. In that sense it is SSL accelerator because if channel is terminated here, it is possible to place an HSM and then get the additional benefit that the traffic comes in clear after the TLS channel is terminated. This permit to perform content inspection.
- **Load balancer**
- **Web accelerator** *(=cache for static content)*
- **Compression**
- **Spoon feeding**
  - Gets from the server a whole dynamic page and feeds it to the client according to its speed, so unloading the application server;

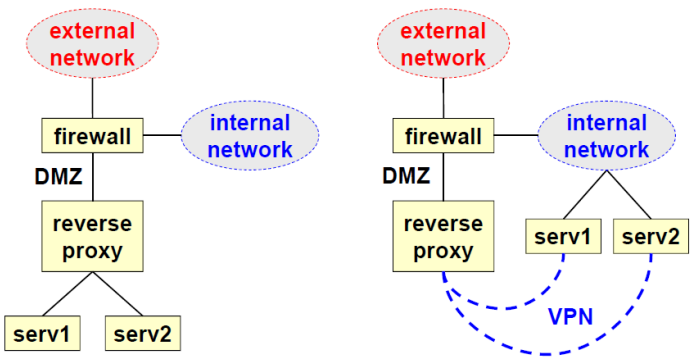


Figure 73 Possible configurations of three-legged firewall when using a reverse proxy. The one on the left is the suggested one.

There are two possible configurations for reverse proxy: the best solution is that, if conceptually a three-legged firewall is used, the reverse proxy should be placed on the DMZ because it will be the public interface, and then it is possible to place behind it the equivalent servers (that are the application servers being accessed from the external users). The problem is in case the application server needs to have access to data located inside the internal network: in this case the server should pass back from the proxy, traverse the firewall and finally enter the internal network.

To handle this case avoiding what just explained, an alternative solution is also possible: the reverse proxy is on DMZ and then a VPN connection is established between the reverse proxy and the servers, which are in the internal network. This should limit the risks because there is no direct access to those servers, but only through the reverse proxy. However, the first solution is the suggested one, since in that case an attack against the (public) servers is confined to the DMZ.

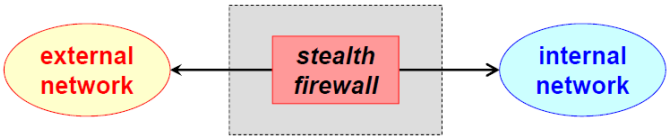
WAF (Web Application Firewall)

Since nowadays web applications are largely used, there is an increasing number of threats. WAF is a module installed at a proxy (forward and/or reverse) to **filter the application traffic**. It checks:

- HTTP commands
- Header of HTTP request/response
- Content of HTTP request/response

The most widely known and used WAF is **ModSecurity** (opensource project) which is a plugin for *Apache* and *NGINX* (which represent 50% and 30% of worldwide HTTP servers). ModSecurity is so popular that OWASP (*Open Web Application Security Project*) has developed a specific **ModSecurity Core Rule Set** (*CRS*). This means that OWASP studied which are the most frequent and well-known attacks and they written a set of rules that permit to ModSecurity to detect and drop those attacks.

Stealth firewall



It is a firewall **without an IP address**, so that it cannot be directly attacked. If a firewall does not have an IP address, there is the problem that firewall cannot be the end point of any communication. But in this case, the stealth firewall will physically intercept packets by setting the interfaces in **promiscuous mode** (mode in which the network card gets a copy of any packets that reach the interface). Then, based on a logic the task of the firewall will be to **copy or discard** network traffic (based upon its security policy) but does not alter it in any way. If the firewall is not addressed in the network is not possible to make remote maintenance/configuration unless another network card is attached to the firewall but connected to dedicated security management network.

Local/personal firewall

Until now we discussed about network firewall, which is a filter that is sitting between two networks (internal and external), but this kind of firewall has a problem: HTTPS, which means that channels are encrypted. It is difficult to look inside packets.

This led to move the firewall where is possible to inspect traffic. In HTTPS traffic is encrypted on client and server. This means that firewall should be moved at client or server: **local** (if installed on the server) **or personal** (if installed on the client) **firewall**. It is directly **installed at the node to be protected**. It protects a single node instead of a whole network. It is typically a **packet filter** but with respect to a normal network firewall, it may **limit the processes** that are allowed:

- *To open network channels towards other nodes (i.e., act as a client)*
- *To answer network requests (i.e., act as a server)*

For example, this firewall can limit traffic going to an unknown process because maybe it could be a malware on the client that is trying to get some information.

It is important to limit the diffusion of malware and trojans, or to detect plain configuration mistakes (for example some services installed by default on the operating system that send or receive data). Firewall management must be separated from system management, otherwise the people installing a service will also activate firewall to permit that kind of traffic (which should be decided by the security manager).

Protection offered by a firewall

The picture displays the fact that in those places of the wall where the bricks are removed to permit traffic, that is the part in which the fire can get into the system. For all channels that have been enabled through the firewall, other protections are needed:

- *VPN;*
- *"semantic" firewall / IDS;*
- *Application-level security.*

Intrusion Detection System (IDS)

IDS is another major component of any security solution nowadays: it is a **system to identify actors using a computer or a network without authorization**. Depending on the feature, it can be extended to **identify authorized actors violating their privileges.**

IDS is based on an important hypothesis: **the behavioural "pattern" of non-authorized users differs from that of the authorized ones**.

An IDS can base its behaviour on two strategies:

- **Passive IDS**
  - Tries to identify visible signs of an attack, using:
    - *Cryptographic checksum* (e.g., tripwire) checking for modified files on the server that should not be modified;
    - *Pattern matching* ("attack signature") looking for specific packets which are typical of an attack;
- **Active IDS**
  - Tries to identify unknown attacks or known attacks but before they produce their negative result. It goes through three steps:
    - "*learning*" = accurate statistical analysis of the system behaviour;
    - "*monitoring*" = active statistical info collection of traffic, data, sequences, actions;
    - "*reaction*" = comparison against statistical parameters (reaction when a threshold is exceeded).

An Active IDS has always someone who looks for IDS' statistics. This is needed because statistics may not be accurate. An attack is detected if it significantly diverges from the statistics related to the behaviour of the system, so there is the chance to have undetected attacks but also false positives. This means that human intervention is often needed to check the alarms raised by the IDS, to carefully analyse the problem, and to understand if there is an attack or it is just a false positive, given the strict threshold. Typically, IDS have both active and passive part.

Topological features of IDS:

- **HIDS (host-based IDS),** looks for statistics inside a single node:
  - *Log analysis (OS, service, or application);*
  - *Internal OS monitoring tools;*
- **NIDS (network-based IDS),** looks at the network traffic:
  - *Network traffic monitoring tools.*

HIDS components: SIV and LFM

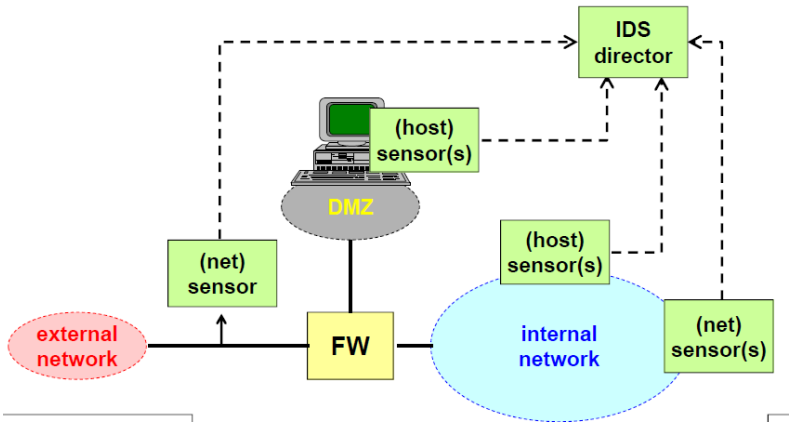
- **System Integrity Verifier**
  - Checks files/filesystems looking for unauthorized changes (e.g., changes to Windows registry, cron configuration, user privileges)
  - e.g., *tripwire*

- **Log File Monitor**
  - Checks the log files (OS and applications) looking for known patterns of successful attacks or attempts (i.e., huge number of failed logins)
  - e.g., *swatch*

NIDS components

- **Sensor** (host/network based):
  - Checks traffic and logs looking for suspect patterns;
  - Then generate the relevant security events;
  - Could also interact with the system (modifying ACLs, performing TCP reset, …)
- **Director:**
  - Coordinates the work of all sensors;
  - Manages the *security database* (e.g., statistics, attack signatures);
- **IDS message system** (to make director and sensors communicate):
  - Provides secure (authentication and integrity) and reliable (nobody should be able to drop a message that was sent by sensors or vice versa) communication among the IDS components. To achieve reliability the communication, if possible, is performed on a separate physical network. If it is not possible, a VLAN or VPN is used.

NIDS architecture



The schema shows how IDS should be configured. Conceptually there is always the three-legged firewall with a connection to the external, one to the internal and one to the DMZ. In the internal network, **sensors** on the most critical **hosts** are inserted (e.g., databases, application servers). There are also **net sensors** needed to check if some malicious traffic succeeding in passing network but is also needed to check the behaviour of users in the internal network.

On the DMZ there is the server reachable from the outside and those are easily attackable. Therefore, a **host sensor** on each server placed on the DMZ should be inserted. The last sensor is the **net sensor** on the external interface of the firewall. Someone think that it is not needed because it is outside but placing it outside of the firewall permit to analyse which are the **attempts**, to get statistics (e.g., firewall blocks 98% of attacks).

On the top right the IDS director collects all the information from sensors.

IPS - Intrusion Prevention System

IPS is a system which is used to **speed-up** and **automate** the reaction to intrusions. It is the pairing of an **IDS** plus a **distributed dynamic firewall**. The IDS send alarms related to a certain type of traffic, then the distributed dynamic firewall when receives an alarm can block all the traffic of that specific kind or isolate a node. This is good if the detection of the intrusion detection system is 100% certain, but if it is not then IPS it is dangerous because it **may take wrong decision and block innocent traffic**. It is a *technology*, not a product, with large impact on many elements of the protection system, which is often integrated in a single product, called IDPS.

Next-Generation Firewall (NGFW)

NGFW is a kind of firewall which **tries to identify applications independently of whatever network port is used**. Imagine that users can only use HTTPS (443/TCP) while all the rest of traffic is blocked. There could be someone that uses port 443 to send not HTTPS but other kind of traffic. With a packet filter is not possible

to detect that. On the contrary, with NGFW is possible to look at the traffic on any port and try to understand what is the real application that is being carried out from that port. If possible, **NGFW try to decipher/re-cipher** the traffic.

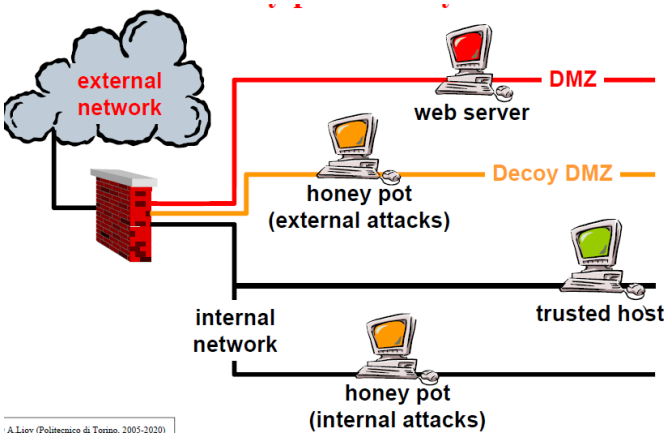
It offers integration with authentication systems such as captive portal, 802.1x, or endpoint authN (Kerberos, Active Directory, LDAP, …), because in this way while normally network firewall is writing policies based on IP addresses, if there is integration with a system that is telling to which user an IP address is assigned, is possible to write **per-user policies** and not for IP addresses. In the same way the fact that is possible to identify applications independent of the port used, is possible to apply also **per-application** policies. Additionally, can also perform filtering based also upon known vulnerabilities, threats, and malware.

Unified Threat Management (UTM)

It is an **integration of several products in a single device** which simplify the management. The device provided is named *UTM appliance/Security appliance* which can contain *firewall, VPN, anti-malware, content-inspection, IDPS, …*

The actual capabilities depend upon the manufacturer. It is mainly targeted to reduce the number of different systems, hence the management complexity and the cost.

Honey pot/Honey net



Honey pot is a way to **attract attackers**. The external network is protected with a firewall, which contains the real DMZ, but additionally there is a second DMZ called **decoy DMZ** where there is a **honey pot**. It is an **easily attackable machine**. The point is to attract attackers and then to have an inspection system that monitors what the attacker is doing, to **understand the behaviour** of the attackers and of new kind of attacks, or to collect malwares. Honey pot are placed also all over the world from antimalware manufacturers just to collect new malwares.

The honey pot can also be placed inside the internal network, to detect if there are some internal users trying to attack the system. For example, is possible to create a fake duplicate salary server just to check if someone is trying to increase its own salary or to read someone else's salary.

E-mail security

MHS (Message Handling System)

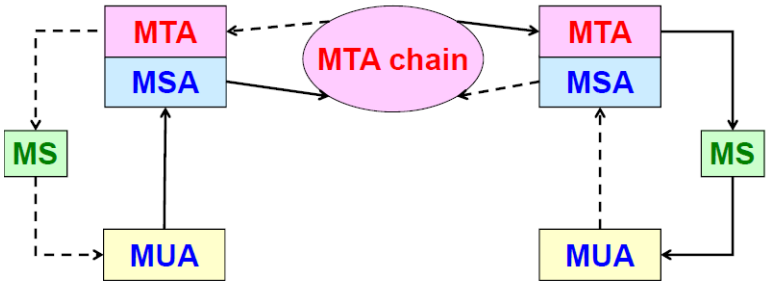


Figure 74 General schema of how the e-mail system works

There is a worldwide system which follow the MHS which contains different component listed in the picture.

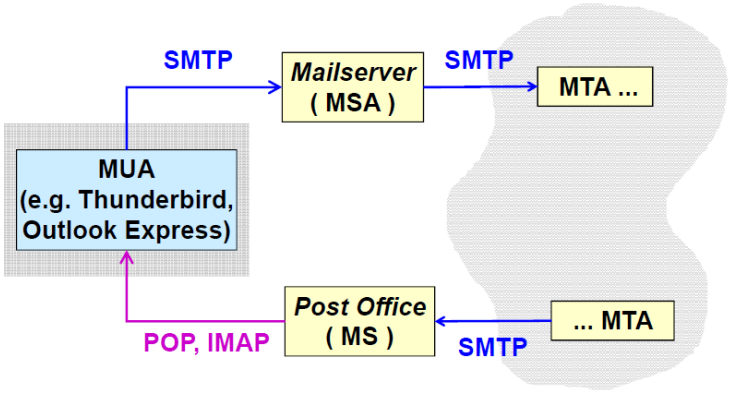
The **MUA (Message User Agent)** is the software component that the user uses to send mails. The mail is then transmitted from the MUA to the **MSA (Message Submission Agent)**, which is another software component that has the task to inject the mail in the mail transport system. The mail transport system is composed of **MTA**s **(Message Transfer Agent)** that are arranged in a **chain**: the mail is given to the next MTA and so on (such as physical office mailing delivery). The chain ends when the message is received at the final destination, which is not the computer or device of the individual

user, but is another server named **MS (Message Store)**. Finally, the destination will use the MUA to read the mail when wanted.

If destination wants to send a reply, it must follow a similar path, but the reverse MSA is different from the sending MSA, the MTA chain may be different, and so on.

There are two ways for interfacing with a mail. The first way is the **e-mail in client-server mode.**

E-mail in client-server mode



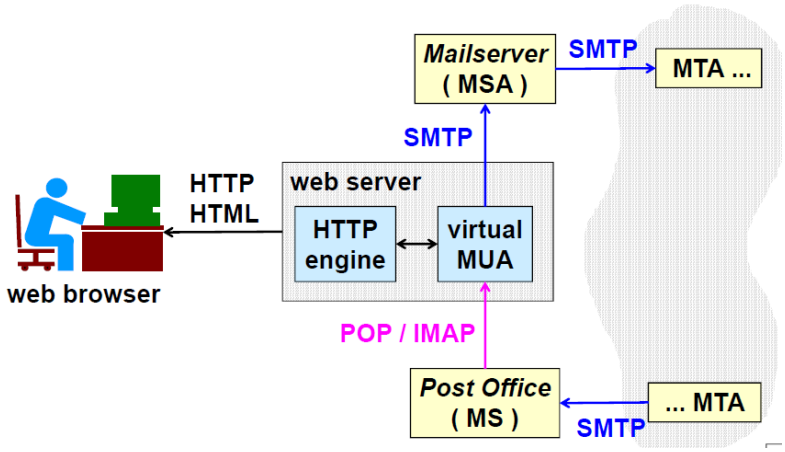
It means that user has got a specific program (i.e., Thunderbird, Outlook) which is configured to know which is the mail server or outgoing **mail server** (MSA). It will use the SMTP protocol to send the mail. This would permit to send a message and then switch off the device, and yet the mail has been submitted. That does not mean that mail has been received because the MSA will continue to send when the network will permit and when the receiving MTA will have space, because each MTA is storing locally the received mail and forward it to the next element in the chain when possible. Even if there is a **network disruption**, *mail will be delivered.* Mails can take up to *three days* to be delivered.

When someone is sending a message, he will inject it through the MTA chain and finally the incoming message will be stored in the **Post Office** (*MS, incoming Mail Server*).

Note that SMTP is a **push protocol**, because mail is always sent to the next item, but when the mail is received at the post office there is no push, but another protocol (**POP/IMAP**) is used when a user wants to check the e-mail, asking to the post office for incoming messages. Here is possible to choose to leave a copy on the MS or delete it. From the point of view of basic security, this is the best solution: the received mail is deleted from the post office. Of course, the various MS may have made an illegal copy of the mail.

Nowadays this schema is not used anymore, the most common is the **webmail schema**.

Webmail



The schema is actually the same with a big difference: no MUA but **web browser** (or mobile app). The real MUA is a **virtual MUA**, which is placed inside the **web server**. In front of the MUA an **HTTP engine** is placed, which permits to write email with standard web formats (HTTP protocol, HTML language, etc.). The point is that the email is stored at the mail server provider. From the point of view of privacy *this is not good*. The reason of why there is unlimited space for mails is because companies have control of mails.

Protocols, ports, and formats

Several protocols, ports and formats are involved:

- **SMTP (Simple Mail Transfer Protocol)**

    - *25/TCP (MTA)*
    - *587/TCP (MSA) when submitting mails*
- **POP (Post Office Protocol)**
    - *110/TCP*
- **IMAP (Internet Message Access Protocol)**
    - *143/TCP*
- **"RFC-822"**
    - *Message format (pure text body).*
- **MIME**
    - *Multimedia extension of RFC-822*

RFC-822 messages

It provides **pure text** mails with only US-ASCII characters on 7 bits, because the 8th can be used for a parity check (residual thing of when networks were unreliable). Lines must be terminated by <**CR**> <**LF**> to make mails be system independent. Messages are composed by header and body:

- **Header**
    - Keywords at the beginning of the line
    - Continuation lines start with a space
- **Body**
    - Separated from the header by an empty line
    - Contains the message

Header RFC-822

Note that even if an interface is used in the own language, that is just an interface because the real messages sent in the network have headers **written in English**.

- **From** or **Sender** is identifying who is sending the message: **from** is the **logical sender**, while **sender** is the **operational sender**. Typically, they are coincident, but sometimes they are not. For example, when sending mails with Polytechnic's system they are the same, but if Gmail is used, there is the option to send mail with Gmail account, but it is possible to label it with a different "From".
- Then there is the **organization, to and subject**.
- **Date** is the time **declared by MUA,** which means that is easy to fake date in a message.
- **Received** corresponds to any MTA that has been traversed. Any MTA puts information in it.
- **Message-id, CC (Carbon Copies)**
- **BCC (Blind Carbon Copies)** that has hidden recipient.
- **Return-Receipt-to** when the sender would like to receive an ack.

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

An SMTP/RFC-822 example

Since SMTP is a text-based protocol and RFC-822 is a text-based format, is possible to send email with keyboard using **telnet**, which opens text-based terminal with remote servers.

```
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
```

- *telnet duke.colorado.edu 25*
  - opens a connection to port 25 (that is where server is listening)
  - The server answer back with a positive response (220).
- *HELO Leonardo.polito.it*
  - All commands in STMP are 4-characters long, so HELO is not a mistake.
  - Server answers back with a positive response
- *MAIL FROM*: *cat*
  - Server accepts because it does not know who is on the side.
- *RCPT TO: franz*
  - Destination server checks if franz it is a user that has got a mailbox on that system. If it is a positive answer, it means *recipient ok.*
- DATA
  - The server answer with 354, which is a positive intermediate.

Then, the message starts following the RFC-822 and then the message is finished with a dot on the last line, as following on the right.

```
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
```

The server answer back with a *250 Ok*, which means that the message has been placed inside the mailbox of user *franz@duke.colorado.edu.* Then with *QUIT* channel is closed.

This has been showed to know that there are a lot of opportunity to create fake mails, because in "From" is possible to write anything. There's no check that the FROM/RCPT TO declared before correspond in the one put in the content of the mail message.

Problems in securing e-mail

- **Connectionless system** (*store-and-forward, also because of MX records*)
- **Untrusted MTAs**
- **Security of MS**
  - Because message is stored in MS
- **Mailing-list encryption**
  - Public key of destination is needed. If emails are sent to mailing-list is not possible to encrypt because there are a lot of emails inside.
- **Compatibility with what is already installed**
- **Concurrent solutions** have been developed in the past**:**
  - *Internet = PGP, PEM, MOSS, S/MIME*
  - *OSI = X.400*
  - *Nowadays the only 2 survivors are: S/MIME and PGP*

Mail spamming

Also named **UBE** (*Unsolicited Bulk Email*) or **UCE** (*Unsolicited Commercial E-mail*) is used for **sending of unwanted messages**:

- ***Unauthorized advertisement***
- ***Attacks*** *(malware, phishing, …)*

Today it is nearly 50% of the total e-mail traffic (54% at March '20, down from 69% in 2012), which means:

- Heavy load on servers and network channels (because of its store and forward nature)
- Heavy annoyance to the users

The term "spam" comes from a sketch from Monty Python related to Canned pork meat. Due to the origin, the opposite of "spam" is "ham" (term used by identification and filtering applications).

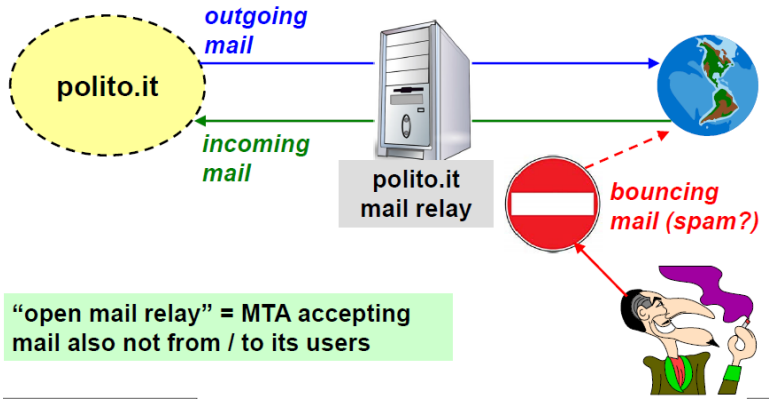
Spamming strategies

The spammer tries to **hide** the real sender but using a **valid sender** because in such a way it is possible for the victim to trust the mail as a good one. Spammer also sends spam via special MTA which are named **open mail relay** because they are MTAs not well configured which accept email from anybody and from any destination. Spammers also use **zombie or botnet** with variable or **phantom IP address** (use addresses that should not be assigned).

Another frequent spamming strategy is the **content obfuscation** (because of programs that check the mail's content to discard email containing keywords associated with spam)**:**

- **Deliberate mistakes** (e.g., Vi@gr@);
- **Image rather than text** (image which contains text, so that it goes undetected by antispamming software, which usually cannot perform text recognition);
- **Bayesian poisoning** (e.g., text from a book): antispam systems usually check the frequency of words compared to standard messages. To poison those detection systems, spammers insert short text in the beginning, which is the real message, and then they insert 20-30 lines for example from Shakespeare.
- **Inside an error message:** in electronic mail there is no guarantee that email is sent to the destination, and if after 3 days the MTA is unable to forward your message it sends back an error. So, spammers send fake error messages in the hope that they are not checked by antispamming software.

(Open) mail relay



Normally, when there is a domain there is also a *domain relay*, composed of two servers (or one including both): it is configured to accept **outgoing mai**l only if they come from the domain "*polito.it*" and it accept incoming mail only if the final destination is "*polito.it*". On the contrary, if there is someone sending an email to the Polito's mail relay, but the destination is in some other places in the world, it is forbidden. This means that the mail relay is not an open mail relay.

If the system is misconfigured and those restrictions are not accepted, then what would happen is that an external user could ask to the Polito mail relay to send email to the outside. However, this restriction must consider the case in which an external valid Polito's user wants to use the service. For this reason, the mail relay must be able to distinguish real from unknown or untrusted users, and of course it is not possible to use the IP address to identify if someone is from polito.it domain.

Anti-spam for MSA

The guideline is **not to configure your own MSA as an "open relay"** but restrict its use only to authorized users, which must be **authenticated**. To authenticate the users of our MSA it is possible to check for:

- **IP address of the MUA**
  - It is a problem with mobile nodes, or because of IP spoofing and malware installed on a valid node.
- **Value of the field From**
  - Can be easily tricked with a fake email
- **SMTP authentication**
  - It is the most secure, but which are the secure authentication methods?

Anti-spam for incoming MTA

DNSBL lists

One would also identify incoming spamming, when receiving mails. The strategy is based on checking if the MTA which is contacting the MS is a well-known spammer, **checking** it against a blacklist or whitelist. A possibility is to check in a **DNSBL** (*DNS-based BlackList*) which works as following:

- A request from MTA with address A.B.C.D is received.
- Is the address A.B.C.D used to send spam?
- *nslookup –q=A D.C.B.A.dnsbl.antispam.net*
- If NXDOMAIN ("*no such domain*") then it is **not** a spammer
- Else the query returns:
  - An address 127.0.0.X (where X is a code providing the reason for being black-listed);
  - A TXT record with more information.
- RFC-5782 "DNS blacklists and whitelists"

Since spammers change MTA quite frequently, maybe those lists are often late in detecting spamming MTAs, so another strategy is not to look at the address of MTA but at the **content of the message**. If message contains a URI (such as for phishing), then a lookup about reputation of that URI is performed using **URI DNSBL** (*URI reputation data*). Many honeypot/spamtrap are placed for capturing spam and classifying the URI found in the messages.

There are several lists (free/commercial, anonymous or not) such as:

- *MAPS RBL (Realtime Blackhole List)*
- *Spamhaus SBL (Spamhaus Block List)*
- *SORBS (Spam and Open Relay Blocking System)*
- *APEWS (Anonymous Postmaster Early Warning System)*

Lists are managed by people that are not easy to be contacted, so URI are not easy to be removed once inserted: it is strongly suggested to correctly configure the MTA. Activate/use the address *abuse@domain*, as required by RFC-2142.

Greylisting

Greylisting is based on the idea that spammers have scarce time, because they are sending the same message to some millions of target users. Spammers would like to send messages as soon as possible, so greylisting means that when someone is contacted from MTA, a *temporary error ("try later")* is given out, and the address of the MTA is kept in memory. If the same MTA comes back within a certain time interval (e.g., 5 minutes) than the connection is accepted. This is based on the fact that typically a spammer does not want to waste time and will skip the victim if receiving an error. This solution delays also the valid mails, and the server is more stressed since it must keep a history of contacts.

If is not possible to have that kind of load on the incoming mail server, is possible to perform **Nolisting** (*poor man's greylisting*). It is based on the same assumption of greylisting, and it consists in more than one mail exchanger (node designed inside DNS as a contact point to receive mail) for the domain, then giving them different priority. In this case the strategy is to define at least 3 mail exchangers: the first one, the one with the fastest priority does not answer; the second one is the correct one; the third does not answer. What happens is that the spammers typically try hither the first, thinking that it is the faster, or the last one, thinking it is the backup of the backup, and that it is not well protected. Like in the previous solution, also in this case ham is delayed as well, because starting from the first the ham will contact the different servers and may not get soon the answer.

DKIM (DomainKeys Identified Mail)

It is a strategy in which the sender uses a signature to guarantee:

- The identity of the sender;
- The (partial) integrity of the message.

This is done via a **digital signature**, created by the MSA or outgoing MTA. This signature covers some headers and part of the body and it is verifiable via a public key (typically inserted in the DNS itself). Recalling the example of Polito's mail server, it means that every mail sent out from the mail relay of Polito could contain a signature with which the Polito's MTA says "yes, it is a mail sent by me and I have authenticated the user" and it is trustable. This system is increasingly being used (e.g., Gmail, Yahoo).

The signature permits to discard messages with fake sender and hence supports anti-spam and anti-phishing.

Another solution is **SPF (*Sender Policy Framework*) – RFC 4408** which is a technique where a mail domain **declares which are its outgoing MTA,** via a specific record in the DNS.

```
$ 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"
```

Performing a NSLOOKUP on polito.it the answer is "*v=spf1 ptr ~all*" which means that *polito.it* is using SPF version 1, any node inside Polito which has got a valid reverse address (so for which the PTR record exist) is a valid sender of electronic mail and "~all" means that everything should be accepted.

On the contrary, for gmail.com the same query is returning "*v=spf1 redirect=_spf.google.com*" and it is telling that the exact list is provided through a redirect and should look into the DNS for "*_spf.google.com*". When the second query is done, another answer is given, that is the list of all nodes in the world that are outgoing mail servers for google. A mail coming from a Google user could come from any of those addresses; if is not one of those addresses, it is possible to reject messages.

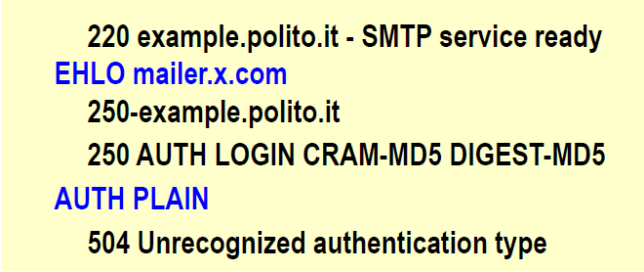
ESMTP

Extended SMTP is defined in RFC-1869 and subsequently incorporated (with SMTP) in RFC-2821. The base protocol and the communication channel are the same. The ESMTP clients must **identify themselves** to the communicating parties with:

- ***EHLO hostname***

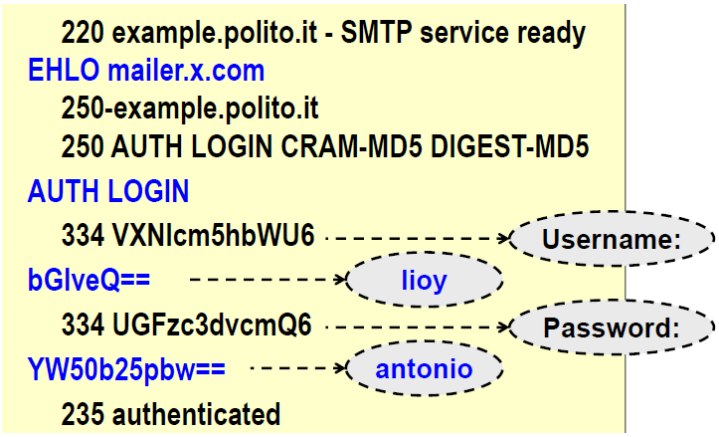
If the receiving server speaks ESMTP, it must **declare the extensions** that it supports, one per line, in its response to EHLO. There are several extensions defined in RFC-4954: the authenticated one add the **command AUTH** plus options of MAIL FROM. They are **needed to authenticate a client before accepting messages from it**. It is useful against spamming because after the EHLO command the server sends the authentication mechanisms supported, the client chooses one and the authentication protocol is executed. If the authentication fails, the communication channel is closed.

Negative AUTH example



In the proposed case the mailer does not know (or does not accept) the authentication method proposed by the client. In black there are the greetings from the server, while in blue the commands from the client. The client says that it is talking the extended protocol (EHLO) and name is *mailer.x.com* and the server reply with his identity. Then the server communicates the three possible authentications: login, CRAM-MD5 or DIGEST-MD5. The client answers with *PLAIN* method, which is not supported by the server and then the client will get an error from server.

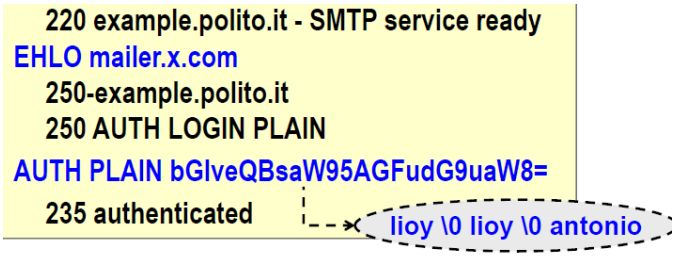
AUTH: LOGIN method



Again, the server gives the list of available authentication methods and in this case the client uses LOGIN method (which is a valid one). The code 3XX is a positive intermediate which is a question. The client gives answers, then another question and then another answer. At the end, client has been authenticated. Strings are just base64 encoding, so it is possible to decode them using openssl. The result of the encode is the one provided in the picture.

AUTH: PLAIN method

The PLAIN method is described in RFC-2595 and just avoids sending the authentication parameters in separate messages and only uses a line:



AUTH PLAIN id_pwd$_{BASE64}$

Id_pwd is defined as following:

[authorize_id] \0 authentication_id \0 pwd

It is an optional authorize identifier, followed by null character and then the authentication identifier, which is compulsory, again a null character and finally the password.

AUTH: challenge-response methods

There are other methods (more secure) because both LOGIN and PLAIN are sending password in cleartext, which is just obfuscated with base64 but there is no real protection. On the contrary, CRAM and DIGEST MD5 are symmetric challenge response methods.

CRAM-MD5

- RFC-2195
- Challenge = ***base64(nonce)***
- Response sent by client = ***base64(usr SP hmac-md5(pwd, nonce)$_{LHEX}$)***
    - User + space + hmac-md5 computed over the nonce with a key equal to the password and encoded in lower-case hexadecimal.

DIGEST-MD5

- RFC-2831
- Similar to HTTP/1.1 digest-authentication
- Declared **obsolete** in RFC-6331 (2011) and replaced with **SCRAM**

AUTH: CRAM-MD5 method

Again, server gives the authentication methods available and the user uses CRAM-MD5. Now the challenge is given from the server, then the user sends the response, and the challenge is given in the picture. It contains number which usually refers to the date and time.

Analysis of CRAM-MD5

Advantages:

- *Permits client authentication* (password);
- *Resistant to replay* (challenge = random number + timestamp + **F**ully**Q**ualified**D**omain**N**ame);
- *Resistant to sniffing* (non-invertible hash).

Disadvantages:

- *No server authentication* (but if used over TLS that could be good because it provides server authentication);
- *Cleartext storage of the password*, **unless** the intermediate steps of HMAC are stored (i.e., $K' \oplus opad$ and $K' \oplus ipad$);
- *Dictionary attacks* if a copy of those database is available to attackers
- *Possible MITM* (channel takeover after CRAM) attacks: always possible **after** authentication, which is a problem of any authentication performed only at the opening of the channel. If there is no integrity for each message, there could be a MITM afterwards.

Protection of SMTP with TLS

RFC-2487 provides "***SMTP Service Extension for Secure SMTP over TLS***".

STARTTLS = *option of EHLO and command*

If the negotiation is successful, the **protocol status is reset** (which means it starts again from EHLO and the extensions supported can be different). On the other side if the negotiated security level is *insufficient*:

- The client sends immediately QUIT and closes the connection;
- The server responds to each command with code 554 (refused due to low security), since in SMTP the connection must be terminated always by the client.

```
   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
```

Security services for e-mail messages

When an MTA is talking with another MTA is not important to have a TLS channel because anyway the mail will be in clear on the MTA itself (a TLS protects only against MITM). Rather than protecting channel (which is meaningless) because email is a hop-by-hop service, emails must be protected themselves. In this way the message is protected independent on which MTA it will be stored on. The wanted features are:

- **Integrity (without direct communication)**
  - The message cannot be modified;
- **Authentication**
  - Identifies the sender;
- **Non-repudiation**
  - The sender cannot deny of having sent the mail;
- **Confidentiality (optional)**
  - Messages are not readable both in transit and when stored in the mailbox.

E-mail security – main ideas

- **No modification to the standard MTA**
  - It means that since adding digital signature or encrypting message will result in binary data, the result of the protected message must be **encoded to avoid problems** when passing through gateways (e.g., Internet-Notes) or MTA non 8BITMIME.
- **No modification to the present UA**
  - Inconvenient user interface
- **With modification to the present UA**
  - Better user interface
- **Symmetric algorithms**
  - For the encryption of messages
  - With *message key*
- **Asymmetric algorithms**
  - To encrypt and exchange the symmetric key
  - For digital signature
- Use **public key certificates** (e.g., X.509) for non-repudiation

If all these operations are doing well, then message security is based only on the security of the UA of the recipient, not on the security of MTA which are not trusted.

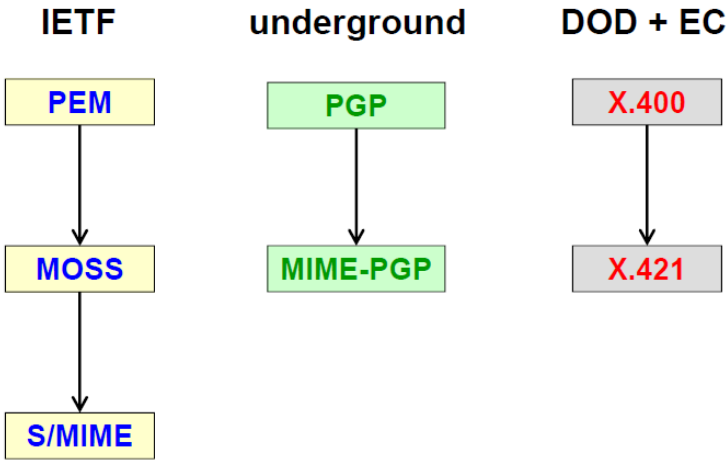
Types of secure messages

- **Clear-signed**
  - *Message in clear* (so that anybody can read it) + *digital signature* (as an attachment or inside the message);
  - Only who has a secure MUA can verify the signature;
  - Risky because if the text-part is not well-encoded, then it might get changes during transmission and signature check will not match. It has to do with the fact that uses only 7 bits for encoding a character, and the most significant one can be cleared by any MTA: this would cause the signature mismatch even if the message is "correct";
- **Signed**
  - [message + digital signature] encoded together (e.g., base64, uuencode)
  - Only who has a secure MUA (or performs operations manually) can decode and verify the signature
- **Encrypted/enveloped**
  - [encrypted message + encrypted keys] encoded
  - Only who has a secure MUA (and the keys!) can decrypt the message
- **Signed and enveloped**

Secure messages: creation

- **Transform message in canonical form**
  - Means standard format, independent from OS/host/net;
- **MIC** (*Message Integrity Code*)
  - Provides integrity and authentication;
  - Typically: message + $(hash(msg))_{privateSenderKey}$.
- **Encryption**
  - Confidentiality
  - Typically: $(message)_{MessageKey}$+$(MessageKey)_{PublicReceiverKey}$
  - Note that the second part may appears multiple time for multiple receivers.
- **Encoding**
- To avoid modification by the MTA
- Typically: *base64*, *uuencode*, *binhex*

Secure electronic mail formats



In the internet world, the *IETF* has first created standard **PEM** which protected just text messages, then **MOSS** that was protecting MIME-based messages and finally the last standard which is nowadays used is **S/MIME** (Secure-MIME). Someone does not trust CAs, so created first the **PGP** standard and then **MIME-PGP** to apply it to MIME messages and it is typical of the underground world. The Department of Defense and part of European Commission for several years used the **X.400** mail system, that was the standard system for OSI system with an extension **X.421** for multimedia. Those systems are now no more used and even governments use S/MIME.

PGP (Pretty Good Privacy)

PGP is a system for **authentication, integrity and confidentiality** for electronic mail or private files. It has the same objectives as PEM and similar organization but less structured. It is peculiar because it does not use Certification Authority but uses the concept of **trusted "friends" and trust propagation algebra**. It is described in RFC-1991 (informational) and RFC-4880 (OpenPGP). It is available for **any kind** of system platform (e.g., UNIX, VMS, MS-DOS, Mac, Amiga, …). The author is Phil Zimmerman and the program have become a symbol of the freedom in Internet.

Phil Zimmermann created the software on behalf of some friends and then released it as freeware in 1991. Before 2000 there was a law in USA that said that was not possible to export strong encryption systems. PGP systems appeared outside of the USA, so the US government decided to punish Zimmermann and accused him of illegal export of encryption system. He has been jailed but then people started complaining then he was released on bail, but investigation and accusation continues until 1996 when accusations are dropped, and Zimmermann creates its own PGP company. This PGP Inc. started a commercial version of PGP that later has been acquired by NAI. In Aug '02 some people examined the source code of PGP and discovered that there was a backdoor that permitted NAI to decrypt all the messages. As a result, Zimmermann was accused to have accepted some requirement for this kind of decryption. In this way he became no more a symbol of freedom for Internet, so he left NAI and created a PGP Co. which is *no more open-source* and *not available for Linux*.

PGP – certification

Since there is no trust in CA, PGP performs certification in the following way: there is a public key of a user which is signed by all people that trust that user. Then the trust is propagated transitively, with some approximation:

- *Completely*
- *Partially*
- *Untrusted*
- *Unknown*

Then, the trust algebra says that to make a partial trust, at least three other partial trusted are needed. The point is to have more friends and to define clearly which is the trust level that user has in them.

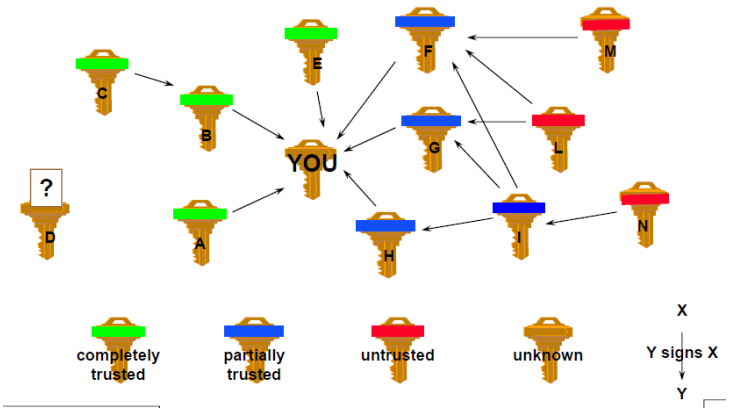
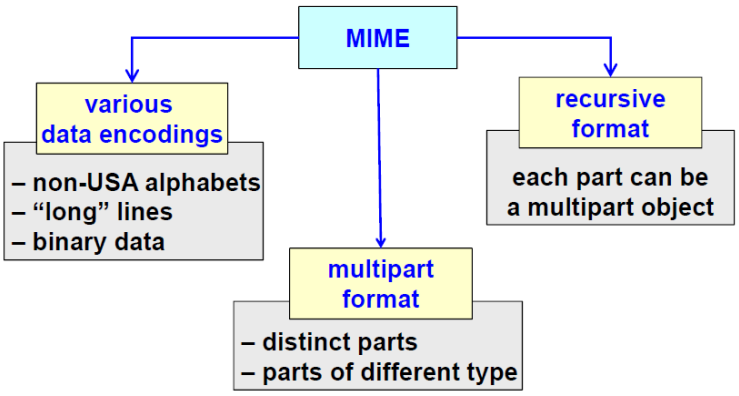


Figure 75 PGP trust propagation: in the example "YOU" completely trusts A, B and E, so C is trusted because B is trusted. F, G and H are partially trusted by "YOU", so I who is partially trusted by three partially trusted peers is partially trusted. On the contrary, L, M and N have less then three partial trust, so they are untrusted. D is unknown.

PGP – key distribution

Public-keys are stored individually by each user (in its own **key-ring**). Then, keys are distributed directly by the owner (at a PGP party) or by a key-server (http, smtp, finger) which is only a distribution point in which the key with all the attached signatures is put. The trust on the key is not based on the fact that the key is being retrieved from the key server, but it is based on the fact that there should be a check if among all the signers there is someone trusted that can prove that the key is trustable. Key distribution is possible also via X.500 or DNS.

MIME (Multipurpose Internet Mail Extensions)



MIME is used to protect e-mail when they contain also various kind of data (attachments, pictures, gifs, …). It solves three problems:

1. It is possible to use **different data encodings** and it is possible to use non-USA alphabets, "long" lines (where "long" means greater than 56 characters, that is the original maximum line length) and binary data (images, films, apps);
2. MIME is also a **multipart format** which means that a message can contain different parts where each part may have a different type;
3. MIME is also a **recursive format** which means that any part of MIME can itself be in multipart object.

Especially the last two features are used in security to perform **digital signature and encryption**, while the support for various data encodings is important because to protect an e-mail with digital signature it is needed to perform the same digest through a hash algorithm. It means that data being transmitted must not be changed (but during transmission MTA has the right to clear MSB): an appropriate encoding can avoid such a manipulation.

Secure multimedia electronic mail (MOSS o S-MIME)

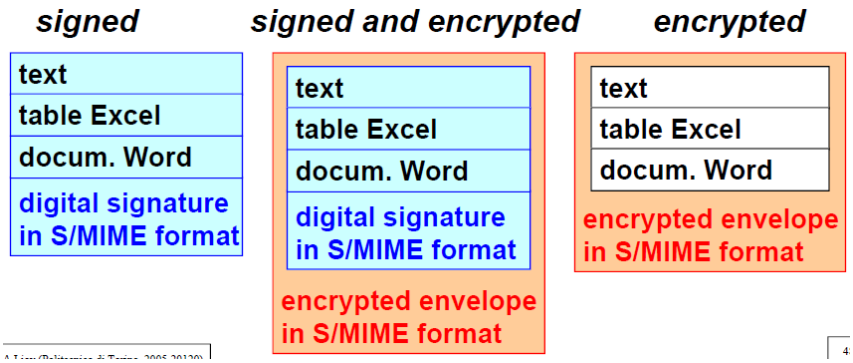


Figure 76 Various ways in which an original MIME object can be modified to have it signed (first from left), encrypted (first on right) or both signed and encrypted (in the middle)

Both standards provide digital signature/encryption with X.509 certificates to protect MIME messages. To create a signed mail messages, a trailing part (that must be in the last position) is added by computing the encoding of the digital signature computed over all the previous parts inside the message. To encrypt the message the original MIME message is considered as a single object and thanks to the recursive property of MIME an encrypted object is created: the message being encrypted can be a signed message, so it is possible to have the original MIME object being signed and encrypted.

RFC-1847

It is a MIME extension for message security (original MOSS standard) to apply digital signature.

```
Content-Type: multipart/signed;
      protocol="TYPE/STYPE";
      micalg="...";
      boundary="..."
```

That required to add a new **Content-Type** to the message that was *multipart/signed* with the specification of the **protocol** being used (protocol is improperly used because it is not a network protocol, but it is the kind of digital signature being applied to the message) and then the ***micalg*** (algorithm used to compute the message integrity code) and as usual the boundary which is the string separating two parts of MIME.

When the encoding described in RFC-1847 is applied then there are *N* body parts where the first **N-1 ones** are those to be protected (e.g., *content-type: image*) and the last one **contains** the **digital signature** (content-type: TYPE/STYPE).

S/MIME

Starting from the original definition of MOSS (RFC-1847), S/MIME was defined and for some time MOSS and S/MIME were competitors (because MOSS was originally defined by the ITF while S/MIME was a product of RSA which is a private company). S/MIME v1 was used only by RSA while S/MIME v2 was published as a series of informational RFC:

- RFC-2311 "*S/MIME v2 message specification*"
- RFC-2312 "*S/MIME v2 certificate handling*"
- RFC-2313 "*PKCS-1: RSA encryption v.1-5*"
- RFC-2314 "*PKCS-10: certification request syntax v.1-5*"
- RFC-2315 "*PKCS-7: cryptographic message syntax v. 1-5*"

Note: the fact that something is published as RFC does not mean that it has been accepted or endorsed by the internet community.

S/MIME v3, v4

S/MIME evolved and v3 and v4 have been created together with internet community. It is no more a set of informational RFCs but the document S/MIME v3 and v4 are *proposed standards*, which means that they are endorsed by internet community.

S/MIME v3 (jun '99) then v3.1 (jul '04) and v3.2 (jan '10)

- *RFC-2633, "S/MIME v3 message specification"*

- *RFC-2632, "S/MIME v3 certificate handling"*
- *RFC-2634, "Enhanced Security Services for S/MIME"*

S/MIME v4 (apr '19)

- *RFC-8551, "S/MIME v4 message specification"*
- *RFC-8550, "S/MIME v4 certificate handling"*

S/MIME architecture

From the architectural point of view a format to represent the protected body of the message is needed. It was:

- **PKCS-7** (on S/MIME v2);
- **CMS** (on S/MIME v3): Cryptographic Message Syntax. It is the evolution of PKCS-7 inside the ITF;
- PKCS and CMS both specifies the cryptographic characteristics and the message types (equivalent to PEM);
- **PKCS-10**: to perform a certificate request through and email message;
- **X.509**: format of public key certificates.

S/MIME v4.0 – algorithms

Digital signature:

- (MUST) ECDSA with curve P-256 and SHA-256
- (MUST) **EdDSA** with curve 25519
- (DISCOURAGED) RSA with SHA-256
- (SHOULD) RSASSA-PSS with SHA-256

Key exchange:

- (MUST) **ECDH** with curve P-256
- (MUST) ECDH with curve X25519 with HKDF-256
- (MUST SUPPORT BUT DISCOURAGED) RSA encryption: used only for legacy applications
- (SHOULD+) RSAES-OAEP

Confidentiality:

- (MUST) AES-128-**GCM** (confidentiality, authentication, and integrity) and AES-256-GCM;
- (MUST SUPPORT BUT DISCOURAGED) AES-128-CBC: will be dropped in the future;
- (SHOULD+) ChaCha20-Poly1305.

Micalg (depends also upon digital signature):

- SHA-256;
- SHA-512.

S/MIME v3.2 – algorithms

Digital signature:

- (MUST) RSA with SHA-256
- (SHOULD+) DSA with SHA-256
- (SHOULD+) RSASSA-PSS with SHA-256
- (MUST SUPPORT BUT DISCOURAGED) RSA with SHA-1
- (MUST SUPPORT BUT DISCOURAGED) DSA with SHA-1
- (MUST SUPPORT BUT DISCOURAGED) RSA with MD5

Key exchange:

- (MUST) RSA encryption
- (SHOULD+) RSAES-OAEP
- (MUST SUPPORT BUT DISCOURAGED) DHE

Privacy:

- (MUST) AES-128 CBS
- (SHOULD+) AES-192 CBC and AES-256 CBC
- (MUST SUPPORT BUT DISCOURAGED) DES EDE3 CBC

Micalg (depends also upon digital signature):

- MD5
- SHA-1
- SHA-224, SHA-256, SHA-384, SHA-512

MIME type

S/MIME creates new parts in MIME multipart messages. Those parts have specific MIME types, examples of MIME types are:

- **application/pkcs7-mime**, this is used when:
  - The part contains an **encrypted** message (*envelopedData*);
  - The part contains a signed message (*signedData*) addressed only to S/MIME users because it is encoded in base64;
  - Message that contain only a public key (= certificate, in a *degenerate signedData body*)
  - Standard extension: .p7m
  - Always base64-encoded, to avoid any manipulation during the transport;
- **Multipart/signed:**
  - Signed messages addressed also to users not supporting S/MIME;
  - The message is in clear;
  - The last MIME part is the signature (for RFC-1847) and it is base64-encoded;
  - Standard extension for the signature: .p7s
- **Application/pkcs10:**
  - Used to send a certification request to a CA;
  - Base64-encoded

```
Content-Type: multipart/signed;
 protocol="application/pkcs7-signature";
 micalg=sha1;
 boundary="-----aaaaa"

-----aaaaa
Content-Type: text/plain
Content-Transfer-Encoding: 7bit

Hello!
-----aaaaa
Content-Type: application/pkcs7-signature
Content-Transfer-Encoding: base64

MIIN2QasDDSdwe/625dBxgdhdsf76rHfrJe65a4f
fvVSW2Q1eD+SfDs543Sdwe6+25dBxfdER0eDsrs5
```

Figure 77 Example of a message using S/MIME multipart/signed content type.

S/MIME examples

- **Encrypted:**
  - B64(P7_enveloped(msg))
- **Signed (only for S/MIME users):**
  - B64(P7_signed(msg));
- **Signed (for generic users):**
  - MIME(msg)+B64(P7_signed_detached(msg));
- **Signed and encrypted:**
  - B64(P7_enveloped(P7_signed(msg)));
  - B64(P7_signed(P7_enveloped(msg))): this is the preferred option since it is more robust to DoS attacks. In fact, in case the content is big, it is preferrable to check the message integrity as soon as possible, in particular before decrypting it, since it decryption is relatively expensive.

Note: *msg* in the previous notation is the RFC-822 body of the message.

Naming in S/MIME

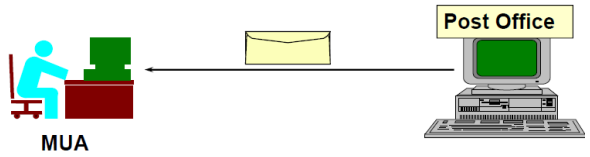
Used for:

- Selecting the certificate;
- Verifying the sender's address: the ***From*** address in the mail must coincide with the address specified in the certificate;
- S/MIMEv2 uses the **Email=** or **E=** fields in the DN of the X.509 certificate, but it is possible to use the extension **subjectAltName** with rfc822 encoding.

S/MIMEv3 mandates the use of the subjectAltName extension with rfc822 encoding: this means that the DN in the certificate must not contain the email address, but it must be in the subjectAltName extension, encoded according rfc822.

Client-server e-mail services

Up to now the protection of the mail message itself has been addressed: with SMTP it is possible to protect outgoing transmission, that is a push operation onto the MSA or the MTA. Recalling the general schema in Figure 74, the pull operation corresponds to checking if there is any email at the post office. To perform this operation, the security properties needed are:



- **authentication of the user**: the mail server must be sure to deliver the emails only to the user they belong to;
- **authentication of the server**: the client must be sure that it is not receiving fake emails from a fake server;
- **confidentiality/integrity of mail messages**:
  - **on the server**: to attain that, this kind of protection must be addressed by the sender, because the receiver has no control on what is stored at the post office;
  - **while in transit**.

So, if the message requires protection the sender should apply S/MIME, and in this way the message is protected both on the servers (the post office itself or even the intermediate MTAs) and in transit.

POP (Post-Office Protocol)

POP has two main versions:

- **POP-2 (RFC-937):** no more used, since did not even used authentication by means of a password, but just the declaration of the username;
- **POP-3 (RFC-1939):** no more used, since it provides user authentication by means of a password in clear;
- **APOP** user authentication (only), by means of a challenge:
  - APOP command replaces the set of commands USER + PASS
  - The challenge is the part of the hello line contained among the parentheses < … > (including the parentheses)
  - Syntax:
    - `APOP user response-to-challenge`
  - Response = MD5(challenge+password), encoded in hexadecimal;
  - Supported by Eudora MUA.

```
    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
```

Figure 78 Example of use of POP3 protocol

```
    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
```

Figure 79 Example of use of APOP protocol

- **K-POP** mutual authentication by means of tickets but requires the use of Kerberos.

IMAP (Internet Mail Access Protocol)

The default user authentication is via username and password sent in clear: however, it supports OTP, Kerberos or GSS-API.

RFC-2595 "Using TLS with IMAP, POP3 and ACAP"

To address the problem of server authentication and the problem of protocols using insecure transmission of username and password, RFC-2595 documents the use of TLS with POP3 and IMAP.

It prescribed that first the communication channel is opened, then the security characteristics are negotiated by means of a dedicated command (in a way similar to what happens with HTTP and SSL/TLS):

- **STARTTLS** for IMAP and ACAMP;
- **STLS** for POP3.

Client and server must allow to be configured to reject username and password if sent in clear. In this way the client compares the identity in the certificate with the identity of the server.

Separate ports for SSL/TLS?

The use of the previous commands to turn a normal channel into a secure one is subject of debate in the security community: the suggested alternative is to use different ports for the secure version of the protocol. For example, in the case of http, it is possible to use the common port 80 and then use the STARTTLS command to make the communication secure, or it is possible to first activate TSL on TCP port 443 and then run http on top of it. Currently, the use of different ports is discouraged by IETF due to the following reasons:

- Involves different URLs (e.g., http and https);
- Involve an incorrect secure/insecure model (e.g., is 40-bit SSL secure? Is it insecure an application without TLS but with SASL?);
- Not easy to implement "use TLS if available";
- Doubles the number of necessary ports.

However, using different ports presents some advantages:

- **Simple to filter traffic on packet-filter firewalls**;
- **TLS with client-authentication allows not to expose the applications to attacks**: this is due to the fact that if the TLS channel is opened before connecting to the application server it is possible to minimize the surface area exposed to an attacker, whose authentication would fail. This is difficult to implement if TLS is activated after, because at that point the attacker would be already connected to the application server (see Authentication in web applications).