Computer and Network Security notes

Salvino Matteo

Contents

1 Introduction 5

1.1 Cryptography vs Security 5
1.2 Symmetric and Asymmetric encryption 6
1.3 Bad ciphers examples 8
1.4 Perfect cipher 9
1.5 Attack models 10

2 Secret key cryptography 11

2.1 DES 12
2.2 Block cipher modes of operation 14
2.2.1 Electronic Codebook 14
2.2.2 Cipher Block Chaining 15
2.2.3 Output Feedback 17
2.2.4 Counter 18
2.3 Internal vs External CBC 18
2.4 AES 19

3 Groups, Rings and Fields 20

3.1 Groups 20
3.1.1 Subgroups 21
3.1.2 Cyclic groups 21
3.2 Rings 21
3.3 Fields 22
3.3.1 Galois Fields 22
3.4 Congruence 23
3.5 Order of Elements 23

4 Data integrity and authentication 24

4.1 Unkeyed hashing functions 26
4.2 Keyed hashing functions 27
4.3 Birthday paradox 28
4.4 Cryptographic hashing function 29
4.5 SHA-1 29
4.6 HMAC 29
4.7 Authenticated Encryption 30

5 Public-Key Cryptography 30
5.1 Diffie-Hellman 31
5.2 RSA 33
5.2.1 RSA algorithm 34
5.2.2 RSA instance 35
5.2.3 Attacks on RSA 35
5.3 Public-key Cryptography Standard 38
5.4 ElGamal Encryption 39

6 Digital signatures 40
6.1 Signature scheme 41
6.1.1 RSA 41
6.1.2 ElGamal 42
6.2 Digital Signature Standard 43

7 Authentication 44
7.1 One way authentication 46
7.2 Two way authentication 47
7.3 Needham-Schroeder protocol 51
7.4 Kerberos protocol 53
7.4.1 Realms 56
7.4.2 v4 vs v5 57
7.5 Public key based 58
7.5.1 Needham-Schoreder 58
7.6 X.509 authentication standard 60
7.7 Public Key Infrastructure 63
7.7.1 X.509 certificates 63
7.7.2 Hierarchy of CAs 64
7.8 Password based 66
7.8.1 Lamport hash 67
7.8.2 Encrypted Key Exchange 68
7.8.3 Simple Password Exponential Key Exchange 69
7.8.4 Password Derived Moduli 69

8 IPSec 71

9 TLS 78

10 Firewalls 83
10.1 Iptables 86

11 Design choices **87**

1 Introduction

The NIST Computer Security Handbook defines the term **computer security** as follows : the protection afforded to an automated information system in order to attain the applicable objectives of preserving the integrity, availability, and confidentiality of information system resources. This definition introduces three key objectives that are at the heart of computer security :

- **Confidentiality** : this term covers two related concepts :
  - **Data confidentiality** : assure that private information is not made available or disclosed to unauthorized individuals.
  - **Privacy** : assures that individuals control what information related to them may be collected and stored and by whom and to whom that information may be disclosed.
- **Integrity** : this term covers two related concepts :
  - **Data integrity** : assures that information 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.
- **Availability** : assures that systems work promptly and service is not denied to authorized users.

These three concepts from what is often referred to as the **CIA triad**. The three concepts embody the fundamental security objectives for both data and for information and computing services.

1.1 Cryptography vs Security

Cryptography is the practice and study of techniques for secure communication in the presence of third parties called adversaries. In general, cryptography is about constructing and analyzing protocols that prevent third parties or the public from reading private messages. His key aspect are data confidentiality, data integrity, authentication and non-repudiation. Cryptography is the intersection of several disciplines like Mathematics, Computer Science, Electrical Engineering, Communication Science and Physics.

Security is the protection of computer systems from the theft of or damage to their hardware, software or electronic data, as well as from the disruption of the services they provide.

1.2 Symmetric and Asymmetric encryption

Symmetric encryption was the only type of encryption in use prior the development of public key encryption in the 1970s. It remains by far the most widely used of the two types of encryption. Before talking about it, we need to introduce some notation : an original message is known as the **plaintext**, while the coded message is called the **ciphertext**. The process of converting from plaintext to ciphertext is known as **encryption**; restoring the plaintext from the ciphertext is known as **decryption**. Such a scheme is known as a **cryptographic system** or a **cipher**. A symmetric encryption scheme has five ingredients :

- **Plaintext** : this is the original message that is fed into the algorithm as input.

- **Encryption algorithm** : it's the encryption algorithm in charge of performing various substitutions and transformation on the plaintext.

- **Secret key** : the secret key is also input to the encryption algorithm and is independent from both the algorithm and plaintext. The algorithm will produce a different output depending on the specific key being used at the time.

- **Ciphertext** : this is the scrambled message produced as output. It depends on the plaintext and the secret key. With two different secret keys, for a given message, the ciphertexts produced will be different.

- **Decryption algorithm** : this is essentially the encryption algorithm run in reverse. It takes the ciphertext and the secret key and produces the original message.

However, there are two fundamental requirements for use in a secure way the conventional encryption mechanism :

1. we need a strong encryption algorithm such that also if the opponents knows it and has access to one or more ciphertexts then he wouldn't b able to decrypt the ciphertext of figure out the secret key.

2. naturally, sender and receiver must exchange copies of the secret key in a secure manner and must keep it secure.

We assume that it's impractical to decrypt a message on the basis of the ciphertext plus the knowledge of the encryption/decryption algorithm. In other words, we don't need to keep the algorithm secret; we need to keep secret only the key. This feature of symmetric encryption is what makes it feasible for widespread use.

Asymmetric encryption is very similar to its symmetric counterpart, but instead of using a single key it uses a pair of keys : **public** and **private** key. It transforms the plaintext into ciphertext using one of the previous key and an encryption algorithm. Using the paired key and a decryption algorithm, the plaintext is recovered from the ciphertext. In fact, the public key as the name suggest is known by anyone and can be used for encrypting a message. In this case, only the receiver with the paired private key (we need to keep it secure) is able to decrypt the message. Viceversa, if the owner of the private key encrypt a message, then it will be possible for anybody to decrypt it using the corresponding paired public key. In the first case we enforce the **confidentiality** measure, whereas in the second scenario we enforce the **authenticity** measure.

To assess effectively the security needs of an organization and to evaluate and choose various security products and policies, we need to make a clear distinction between two important concepts :

- **Threat** : it's a potential for violation of security, which exists when there is a circumstance, capability, action or event that could breach security and cause harm. That is, a threat is a possible danger that might exploit a vulnerability.
- **Attack** : it's an assault on system security that derives from an intelligent threat; that is, an intelligent act that is a deliberate attempt to evade security services and violate the security policy of a system.

Now, suppose that Alice and Bob uses a shared encryption scheme for establishing a reliable communication line in order to sends a message m in a confidential way. In this scenario, the most common threat is the **Man In The Middle (MITM)**, where the adversary can :

- only read the exchanged messages in order to obtain information that is being transmitted. This action is also known as eavesdropping, and it's a **passive attack**. They are very difficult to detect, because they do not involve any alternation of the data. However, we can prevent the success of these attacks, usually by means of encryption.

- modify the messages or send fake messages pretending to be someone else in the network. Since the attacker perform some active action, it's an **active attack**. Some examples of active attacks are : **masquerading**, where the attacker pretends to be somebody else, **replay**, which involves the passive capture of information and its subsequent retransmission to produce an unauthorized effect, or **Denial of Service (DoS)**, which prevents the normal use or management of the communication facilities. In this latter case, are often used SYN packets, because when the server receives a SYN, it opens a new connection and create a new data structure (until it's full). It's quite difficult to prevent active attacks because of the wide variety of potential physical, software and network vulnerabilities. Instead, the goal is to detect active attacks and to recover from any disruption or delays caused by them.

The adversary must not be able to determine any meaningful information about m, even in probabilistic sense. To do this the keys must be unknown to the attacker. In such scenario, it's hard to obtain even partial information on the message or find the key even if the attacker know the clear text. For hard we meant that it's computationally hard for the attacker, i.e. takes long time even if the most powerful computers are available.

1.3 Bad ciphers examples

Caesar cipher is a substitution cipher which replaces each letter of the alphabet with the letter standing three places further down the alphabet. The alphabet is wrapped around, so that the letter following Z is A. If it's known that a given ciphertext is a Caesar cipher, then a brute-force cryptoanalysis is easily performed : simply try all the 25 possible keys. In fact, we can perform a brute-force attack because the encryption and decryption algorithms are known, there are only 25 keys to try and the language of the plaintext is known and easily recognizable. What generally makes this type of attack impractical is the use of an algorithm that employs a large number of keys. The last characteristic is also significant. If the language of the plaintext is unknown, then the plaintext output may not be recognizable. Furthermore, the input may be abbreviated or compressed in some fashion, again making recognition difficult.

Monoalphabetic ciphers : With only 25 possible keys, the Caesar cipher is far from secure. A dramatic increase in the key space can be achieved by allowing an arbitrary substitution. A **permutation** of a finite set of elements S in an ordered sequence of all the elements of S, with each element appearing exactly once. In general there are $n!$ permutations of a set of n elements, because the first element can be chosen in one of n ways, the second in $n-1$ ways, and so on. If the cipher line can be any permutation of the 26 alphabetic characters, then there are 26! possible keys. Such an approach is referred to as a **monoalphabetic substitution cipher**, because a single cipher alphabet is used per message. There is, however, another line of attack. If the cryptanalyst knows the nature of the plaintext, then he can exploit the regularities of the language (letters and digrams frequency analysis).

1.4 Perfect cipher

Given a plaintext space = $\{0,1\}^n$ with D known and a ciphertext C, the probability that exists k_2 such that $D_{k_2}(C) = P$ for any plaintext P is equal to the **apriori probability** that P is the plaintext. In other words, the ciphertext doesn't reveal any information on the plaintext. In formal way, using the conditional probability definition and Bayes theorem, assuming that the plaintext and ciphertext are independent, we have that

$$Pr[plaintext = P|ciphertext = C] = Pr[plaintext = P].$$

A real example of perfect cipher is the**Vernam cipher** also known as **One Time Pad**. It's uses a keyword that is as long as the plaintext and has no statistical relationship to it. The key is used to encrypt and decrypt a single message and then is discarded. It's works on bits rather than letters. Let $\{0,1\}^n$ be the plaintext and key space. The scheme is symmetric, and the key K is chosen at random. The system can be expressed as follows :

$$E_K(P) = C = P \oplus K,$$
$$D_K(C) = C \oplus K = P \oplus K \oplus K = P,$$

We should never use the same key to encrypt two different texts, because it makes possible to xor back the original text, i.e. $P_1 \oplus K = C_1$, $P_2 \oplus K = C_2 \to P_1 \oplus P_2 = C_1 \oplus C_2$. We claim that OTP is a perfect cipher, but the problem here is the size of key space. There is the **Shannon theorem**, which state that a cipher cannot be perfect if the size of its key space is less than the size of its message space.

Proof. Lets prove this theorem by contraction. Thus, we assume that $\#keys < \#message$ and consider a ciphertext C_0 such that $Pr[C_0] > 0$. Now, for some key K, consider $P = D_K(C_0)$. There exists at most $\#keys$ such messages (one for each key). Now, we choose a message P_0 such that it's not in the form of $D_K(C_0)$ (there exists $\#messages - \#keys$ messages). Hence $P[C_0|P_0] = 0$. But in a perfect cipher $P[C_0|P_0] = P[C_0] > 0$. Thus, we have reached a contradiction. □

1.5 Attack models

There are several types of attack that an attacker could perform such as :

- **Eavesdropping** : the attacker secretly listening to private conversation of other without their consent.

- **Known plaintext** : the analyst may be able to capture one or more plaintext messages as well as their encryptions. With this knowledge the analyst may be able to deduce the key on the basis of the way in which the known plaintext is transformed.

- **Chosen plaintext** : the analyst is able somehow to get the source system to insert into the system a message chosen by himself. The analyst choose the message to encrypt, and may deliberately pick patterns that can be expected to reveal the structure of the key.

- **Adaptive chosen plaintext** : the attacker can choose a sequence of plaintexts to be encrypted and have access to the ciphertexts. At each step he has the opportunity to analyze the previous results before choosing the next plaintext. This allow him to have more information when choosing the future plaintexts than the information gained from the previous attack.

- **Chosen ciphertext** : the attacker can choose an arbitrary ciphertext and have access to plaintext decrypted from it. From these pieces of information he can attempt to recover the hidden secret key used for the decryption process.

- **Lunchtime attack** : this is a variant of chosen ciphertext attack, indeed the attacker can only have access to the system for a limited time or a limited number of plaintext-ciphertext pairs, after which he must show progress. The name comes from the common security vulnerability in which an employee signs into his encrypted computer and then leaves it unattended while he goes to lunch, allowing an attacker a limited time access to the system.

- **Adaptive chosen ciphertext** : in this attack the adversary can choose a series of ciphertexts and see the resulting plaintexts, with the opportunity at each step to analyze the previous ciphertext-plaintext pairs before choosing the next ciphertext.

2 Secret key cryptography

Suppose that Alice and Bob communicate using a crypto protocol E with a secret key K. The adversary knows E and some exchanged messages, but ignores K. We can implement this protocol in two ways as a :

Stream cipher : It's uses a secret key also called **seed** as input to a pseudorandom bit generator, which produces a stream of 8-bit numbers that are apparently random. The output of the generator, called a **keystream**, is combined one byte at a time with the plaintext stream using the XOR operation. Its goal is to simulate the One Time Pad. The difference is that a OTP uses a genuine random number stream, whereas a stream cipher uses a pseudorandom number stream. There are some important design consideration to take for a stream cipher :

- the encryption sequence should have a large period. In fact, a pseudorandom generator uses a function that produces a deterministic stream of bits that eventually repeats. The longer the period of repeat is the more difficult will be to do cryptoanalysis.

- the keystream should approximate the properties of a true random number stream as close as possible.

- to guard against brute-force attacks the key needs to be sufficiently long.

With a properly designed pseudorandom number generator, a stream cipher can be as secure as a block cipher of comparable key length. A potential advantage that a stream cipher have is that when it doesn't use block ciphers as a building block, then it is faster and use far less code than do block ciphers. A stream cipher can be **synchronous** or **asynchronous**. It's synchronous when the keystream is generated independently of the plaintext and ciphertext messages. It's asynchronous when uses several of the previous N ciphertext digits to compute the keystream. An example of stream cipher is **RC4**, which was designed in 1987 by Ron Rivest for RSA Security. It's a variable key size stream cipher with byte-oriented operations. The algorithm is based on the use of a random permutation. Analysis shows that he period of the cipher is overwhelmingly likely to be greater than 10^{100}. It's very fast because for each output byte are required only $8 - 16$ instructions. First of all a variable-length key from 1 to 256 bytes is used to initialize a 256-byte state vector S. At all times, S contains a permutation of all 8-bit numbers from 0 to 255. For encryption and decryption a byte k is generated from S by selecting one of the 255 entries in a systematic manner. As each value of k is generated, the entries in S are once again permuted. Once the S vector is initialized, the input key is no longer used. Stream generator involves cycling through all the elements of $S[i]$, and for each $S[i]$, swapping $S[i]$ with another byte in S according to a scheme dictated by the current configuration of S. To encrypt, XOR the value k with the next byte of plaintext. To decrypt, XOR the value k with the next byte of ciphertext. [To deepen] However, it's broken because it's possible to predict the next bit generated.

Block cipher : It's one in which a block of plaintext is treated as a whole and used to produce a ciphertext block of equal length (typical block size of 64 or 128 bits). Using some mode of operation a block cipher can be used to achieve the same effect as a stream cipher. A block cipher operates on a plaintext block of n bits and uses a k of fixed number of bites to produce a ciphertext block of n bits. There are 2^n possible different plaintext blocks and, for the encryption to be reversible, each must produce a unique ciphertext block. The length of both block and key are fixed, and they doesn't need to be necessarily equal.

2.1 DES

DES (Data Encryption Standard) is a symmetric block cipher using 64 bit block and a 56 bit key. It was developed at IBM, approved by the US government (1976) as a standard. The small length of the key makes him too insecure for most current applications. In january 1999, distributed.net and the Electronic Frontier Foundation collaborated to publicly break a DES key in around 22 hours. There are also some analytical results which demonstrate theoretical weaknesses in the cipher, although they are infeasible to mount in practice. The algorithm used by DES is believed to be practically secure in the form of 3DES, although there are theoretical attacks. The 3DES is a symmetric-key block cipher, which applies the DES cipher algorithm three times to each data block using three different keys. The original DES cipher's key size of 56 bits was generally sufficient when that algorithm was designed, but the availability of increasing computational power made brute-force attacks feasible. 3DES provides a relatively simple method of increasing the key size of DES to protect against such attacks, without the need to design a completely new block cipher algorithm. However, by having a deeper look to it, seem easy to brute-force this schema. In fact, we will use two keys instead of three. Notice that we start using the encryptor with key k_1, then we decrypt with another key k_2, so it's not really a decryption. Then we use k_1 for encrypting again. But why we have not considered a double encryption ? Doubling DES or whatever symmetric block cipher is not a good idea because there is an attack called **Meet In The Middle (MITM)** that can be performed. Suppose that we are using an encryption scheme based on 2DES (it will apply the encryption two times). Suppose that we are using 2 keys, k_1 and k_2 so it means that the apparent size of final key is $56 \cdot 2 = 112$. If it happens that we know a plaintext-ciphertext pair, then we can break the encryption. We can get this knowledge about this pair in several cases such as using the chosen ciphertext or chosen plaintext attack. Assume that the size of the key is n bits. So, if we are using two keys in order to make the brute-force attack we should try 2^{2n} keys. Actually, we can break the security of this encryption by using 2^{n+1} encryptions. The adversary prepares what is called a **lookup table** and start computing all possible encryptions, remember that the attacker knows a pair, so he prepares a table, starting from the first key and gets the corresponding 2^n ciphertext. If n is too big this is impossible to perform. Now, after building this table the adversary starts a new phase, he runs again 2^n attempts by computing the decryption of ciphertext C by using all the possible keys. For every key, he computes the decryption and check if it's in the table. All possible encryptions are written in the table, when the adversary tries all possible keys, he tries to guess what is k_2. When finding k_2 he will decrypt the ciphertext obtaining not the original plaintext, but he will get what is inside, i.e. another ciphertext. So, the complexity of this algorithm is $O(2^n)$ steps. Then order of $O(2^n)$ steps but every step require a search. To search in efficient way we need an ordered table using for example an incremental sorting approach, just build a table and then sort it. We can use any sorting algorithm running on time $2^n \cdot \log(2^n)$. This is equal to 2^n times n and this is somewhat requesting an increase of this number of steps. So with one try we attack two keys, this is why we don't use 2DES. The security of 3DES is security of twice the size of the key.

2.2 Block cipher modes of operation

A block cipher takes a fixed-length block of text of length b bits and a key as input and produces a b-bit block of ciphertext. If the amount of plaintext to be encrypted is greater than b bits, then the block cipher can still be used by breaking the plaintext up into b-bit blocks. When multiple plaintext are encrypted using the same key, a number of security issues arise. To apply a block cipher in a variety of applications, five modes of operations have been defined by NIST. In essence a mode of operation is a technique for enhancing the effect of a cryptographic algorithm or adapting the algorithm for an application, such as applying a block cipher to a sequence of data blocks.

2.2.1 Electronic Codebook

Electronic codebook (ECB) is the simplest mode, in which the plaintext is handled one block at a time and each block of plaintext is encrypted using the same key. The term codebook is used because, for a given key, there is a unique ciphertext for every b-bit block of plaintext. For a message longer than b bits, the procedure is simply to break the message into b-bits blocks, padding the last block if necessary. Decryption is performed one block at a time, always using the same key. The ECB method is ideal for a short amount of data, such an encryption key. Thus, if you want to transmit a DES or AES key securely, ECB is the appropriate mode to use. Now, we want to point out the ECB main characteristics as follows :

- **Pattern hiding** : it doesn't hide patterns well, because if the same b-bit block of plaintext appears more than once in the message, it always produce the same ciphertext. For lengthy message, the ECB mode may not be secure. If the message is highly structured, then a cryptanalyst may exploit these regularities. If the message has repetitive elements with a period of repetition a multiple of b bits, then these element can be identified by the analyst. Furthermore, the analyst can also rearrange or substitutes the blocks.

- **Parallelizability** : In most organizations the encryption is made not by a computer program, by an application, small devices, is hardware. More performance with respect to software. If the organization decides to buy 10 encryptors, they can do in parallel some encryption, up to 10. Can we do in parallel the encryption and the decryption ? In this case, the answer is yes for both cases.

- **Preprocessing** : Imagine you are the officer for making encryption and decryption (i.e. availability of encryptors and decryptors). You also know the key. The question is the following : is there any possible task that we can do in advance for making the encryption and decryption process faster ? We don't see any possible preprocessing to make the process of encryption faster. We just use the encryptor, the key and the plaintext is unknown so we cannot preprocess the plaintext.

- **Errors propagation** : Let's imagine we make some encryption and we get a ciphertext with 1 bit wrong. Can we imagine what will be the consequence coming from this error ? The error is just impacting on the block where the error occurred. This idea is good for every symmetric block of encryption algorithm. In general, we can expect that one bit wrong here will make rubbish on the corresponding plaintext. The error is not propagated. Is this good or not ? It depends. An error can occur for two reasons : a transmission error (and we would have an opportunity to recover from this error) and a bit changes from an attacker. The goal of the attacker is to make some sabotage, because he is not able to change bits in a way that the decryption will give another plaintext. The attacker may change 1 or more bit. In this case, the effect of the attack is damaging one block. We consider this property as a bad property because it allows the attacker to make a very precise attack. The error propagation depends on your purposes, from other settings of your environment. The probability of this occurring increases as the file becomes grows.

2.2.2 Cipher Block Chaining

To overcome the security issues of ECB, we would like a technique in which the same plaintext block, if repeated, produces different ciphertext blocks. A simple way to satisfy this requirement is the **Cipher Block Chaining (CBC)** mode. In this scheme, the input to the encryption algorithm is the XOR of the current plaintext block and the preceding ciphertext block. Also in this case, the same key is used for each block. In effect, we have chained together the processing of the sequence of plaintext blocks (i.e. it's an asynchronous stream cipher). So, repeating patterns of b bits are not exposed. As with the ECB mode, the CBC mode requires that the last block be padded to a full b bits if it's a partial block. For decryption, each cipher block is passed through the decryption algorithm. The result is XORed with the preceding ciphertext block to produce the plaintext block.

In formal way, we have

$$C_j = E(K, [C_{j-1} \oplus P_j].$$

Then

$$D(K, C_j) = D(K, E(K, [C_{j-1} \oplus P_j])) = C_{j-1} \oplus P_j \oplus C_{j-1} = P_j.$$

To produce the first block ciphertext, an **initialization vector (IV)** is XORed with the first block of plaintext. On decryption, the IV is XORed with the output of the decryption algorithm to recover the first block of plaintext. The IV must be known to both the sender and receiver but be unpredictable by a third party. In particular, for any given plaintext, it must not be possible to predict the IV that will be associated to the plaintext in advance of the generation of the IV. For maximum security, the IV should be protected against unauthorized changes. This could be done by sending the IV using ECB encryption. In fact, if an opponent is able to fool the receiver into using a different value for IV, then the opponent is able to invert selected bits in the first block of the plaintext. As the previous mode of operations, we want discuss some key points such as :

- **Pattern hiding** : we also said that CBC produces different ciphertext blocks for the same plaintext block. So, it hide patterns very well.

- **Parallelizability** : encryption is totally sequential, because for encrypt one block you need the previous one. Instead, decryption can be done in parallel, because decrypting a block require the knowledge of the current ciphertext block and the previous one, but we have the whole ciphertext, thus the possibility to decrypt different ciphertext blocks at the same time.

- **Preprocessing** : can the officer do some task the day before the encryption or decryption when he is not yet known the plaintext but he knows the key ? We don't see any possibility to get some advantages by spending some time before making encryption or decryption. If you don't known the plaintext you can't start the process.

- **Errors propagation** : it's clear that if you get a one bit error, it will impacting on two blocks. But the way it's impacting is different, one bit error is generating garbage in the corresponding block, one bit error in the next block. Don't forget that the error is just a bit flip. It's possible that if the adversary is having already some information about the plaintext, he may decide to change one bit on the plaintext just by attacking the previous block of ciphertext.

Ciphertext Stealing Ciphertext stealing (CTS) is a general method of using a block cipher mode of operation that allows for processing of messages that are not evenly divisible into blocks without resulting in any expansion of the ciphertext, at the cost of slightly increase of complexity. CTS is a technique for encrypting plaintext using a block cipher, without padding the message to a multiple of the block size, so the ciphertext has the same size as the plaintext. It does this by altering the processing of the last two blocks of the message. The processing of all blocks excepts the last two block is unchanged, because a portion of the second-last block's ciphertext is "stolen" to pad the last plaintext block. The padded final block is then encrypted as usual. Decryption requires decryptingthe final block first, then restoring the stolen ciphertext to the second-last block, which can be decrypted as usual. In principle any block-oriented block cipher mode of operation can be used, but stream-cipher-like modes can already be applied to messages of arbitrary length without padding, so they do not benefit from this technique. The common modes of operation that are coupled with ciphertext stealing are ECB and CBC. CTS for ECB requires the plaintext to be longer than one block. CTS for CBC mode doesn't necessarily requires the plaintext to be longer than one block. In the case where the plaintext is one block long or less, the initialization vector (IV) can act as the prior block of ciphertext. In this scenario, a modified IV must be sent to the receiver. This may not be possible in situations where the IV can not be freely chosen by the sender when the ciphertext is sent, and in this case ciphertext stealing for CBC mode can only occur in plaintexts longer than one block.

2.2.3 Output Feedback

Output Feedback (OFB) mode makes a block cipher into a synchronous stream cipher. The output of each block depends on the key and on the seed. Is it useful that is a synchronous stream cipher ? Yes, because we can apply some preprocessing technique in order to speed up encryption and decryption process. In fact, before making encryption we can compute the key stream in advance, since it doesnt depends from the plaintext. Furthermore, we can parallelize encryption process but only if we have applied some preprocessing technique first. If we want to decrypt we need encryptors again, same used to make encryption. Even in decryption can we have some useful preprocessing ? The answer is yes, because we can compute the keystream just knowing the seed and the key. What is the effect of one bit error ? It is affecting one bit on the plaintext. Is it propagated ? No, just the same effect we have on a pure synchronous stream cipher. It hides also plaintext pattern well, because same plaintext blocks will be XORed with a different encrypted version of the keystream. OFB is a way to implement a good stream cipher starting from a block cipher. Since using OFB we are going back to the typical characteristics of a stream cipher, we need what property ? The property that the adversary is not able to predict the next bits. It is a good practice to start from a secret seed, because the generation of the keystream is a critical point. This means to encrypt the seed if we want to use OFB.

2.2.4 Counter

Counter (CTR) mode is very similar to OFB mode. Instead of computing the keystream by propagating the bits of encryptor to the next encryption, it computes several encryption using a counter. It sets the counter initial value and for each encryption increments its value by one. For encryption, the counter is encrypted and then XORed with the plaintext block to produce the ciphertext block. For decryption, the same sequence of counter values is used, with each encrypted counter XORed wit ha ciphertext block to recover the corresponding plaintext block. Thus, the initial counter value must be made available for decryption. For the last plaintext block, which may be a partial block of u bits, the most significant u bits of the last output block are used for the XOR operation; the remaining $b - u$ bits are discarded. As with the OFB mode, the initial counter value must be a nonce; that is, it must be different for all of the messages encrypted using the same key. In contrary, then the confidentiality of all the plaintext blocks corresponding to the counter value repeated may be compromised. We can see it as a synchronous stream cipher. We can do preprocessing to compute all the bytes needed for computing the XOR with the plaintext and in order to make decryption we have to replicate the computation of the keystream, so we need again encryptors to decrypt. We can perform encryption and decryption in parallel on multiple blocks, since there isn't any concept of chaining. Since the execution of the encryption algorithm doesn't depends on input of the plaintext of ciphertext, then preprocessing can be used to prepare the output of the encryption boxes that feed into the XOR functions. However, we can encrypt or decrypt blocks in a random fashion.

2.3 Internal vs External CBC

Referring to 3DES in CBC mode, we can perform CBC on the outside or inside. External CBC means that each block is triply encrypted, and then CBC is done on the triply encrypted block. Instead, internal CBC would be to completely encrypt the message with k_1 and CBC, then take the result and completely decrypt it with k_2 and CBC, and then take again the result and completely encrypt it with k_1. In CBC it's possible to make a predictable change to plaintext block n, for instance flipping bit x, by flipping bit x in ciphertext block $n-1$. There is a side effect of completely garbling plaintext block $n-1$. With CBC done on the outside, an attacker can still do the same attack. An attacker that flips bit x in ciphertext block $n-1$ will completely and unpredictably garble plaintext block $n-1$. However, plaintext block n will have bit x flipped, and all plaintext blocks other than $n-1$ and n will be unaffected. With CBC done on the inside, any change to ciphertext block n completely and unpredictably alters all plaintext blocks from n to the end of the message. This makes CBC done on the inside more secure. Another advantage of CBC on the inside is the performance. In fact, it's possible to use three times as much hardware and pipeline the encryptions so that it's a as fast as single encryption. With CBC on the outside, this is not possible.

2.4 AES

AES is a symmetric block cipher that is intended to replace DES as the approved standard for a wide range of applications. Compared to public-key ciphers such as RSA, the structure of AES and most symmetric ciphers is quite complex and cannot be explained as easily as many other cryptographic algorithms. In AES, all operations are performed on 8-bit bytes. In particular, the arithmetic operations of addition, multiplication and division are performed over the finite field $GF(2^8)$. The input to the encryption and decryption algorithms is a single 128-bit block. This block is depicted as a 4×4 matrix of bytes. This block is copied into the State array, which is modified at each stage of encryption or decryption. After the final stage, State is copied to an output matrix. The ordering of bytes within a matrix is by column. So, for example, the first four bytes of a 128-bit plaintext input to the encryption cipher occupy the first column of the in matrix, the second four bytes occupy the second column, and so on. As well as the key will be stored in this way. The cipher consists of N rounds, where the number of rounds depends on the key length : 10 rounds for a 128-bit key, 12 rounds for a 192-bit key and 14 rounds for a 256-bit key. The first $N-1$ rounds consist of four distinct transformation function :

1. **SubBytes** : it uses a simple 16×16 lookup table called **S-box**, that contains a permutation of all possible 256 8-bit values. Each individual byte of State is mapped into a new byte in the following way : the

leftmost 4 bits of the byte are used as a row values and the rightmost 4 bits are used as a column value. These row and column serve as indexes into the S-box to select a unique 8-bit output value.

2. **ShiftRows** : the first row of State is not altered. For the second row, a 1-byte circular left shift is performed. For the third row, a 2-byte circular left shift is performed. For the fourth row, a 3-byte circular left shift is performed.

3. **MixColumns** : each byte of a column is mapped into a new value that is a function of all four bytes in that column.

4. **AddRoundKey** : the 128 bits of State are bitwise XORed with the 128 bits of the round key.

The final round contains only three transformation, and there is a initial single transformation (AddRoundKey) before the first round, which can be considered Round 0. Each transformation takes one or more 4×4 matrices as input and produces a 4×4 matrix as output. Also, the key expansion function generates $N+1$ round keys, each of which is a distinct 4×4 matrix. Each round key serve as one of the inputs to the AddRoundKey transformation in each round. The AES key expansion algorithm takes as input a four word (16 bytes) key and produces a linear array of 44 words. This is sufficient to provide a four-word round key for the initial AddRoundKey stage and each of the 10 rounds of the cipher. Breaking 1 or 2 rounds is easy, but it's not known how to break 5 rounds.

3 Groups, Rings and Fields

Groups, rings and fields are the fundamentals elements of a branch of mathematics known as abstract algebra or modern algebra. In abstract algebra, we are concerned with sets on whose elements we can operate algebraically; therefore, we can combine two elements of the set to obtain a third element of the set.

3.1 Groups

A group G, sometimes denoted by (G, +), is a set of elements with a binary operation + (addition) that associates to each ordered pair (a,b) of elements in G an element (a + b) in G, such that the following axioms are obeyed :

- **Closure** : $\forall a, b \in G$ then $a + b \in G$

- **Associative** : $\forall a, b, c \in G,\ (a + b) + c = a + (b + c)$
- **Identity element** : $\exists 0 \in G, \forall a \in G,\ a + 0 = a$
- **Inverse element** : $\forall a \in G, \exists - a \in G, a + (-a) = 0$
- **Commutative** : $\forall a, b \in G, a + b = b + a$

3.1.1 Subgroups

Let (G, +) be a group, (H, +) is a subgroup of (G, +) if it is a group, and $H \subseteq G$.
Lagrange theorem : For any finite group G, the number of elements (order) of every subgroup H of G divides the order of G (i.e. $|H|$ divides $|G|$).

3.1.2 Cyclic groups

In group theory, a cyclic group is a group that is generated by a single element. Therefore, it's a set of invertible elements with a single associative binary operation, and it contains an element g such that every other element of the group may be obtained by repeatedly applying the group operation to g or its inverse. Each element can be written as a power of g in multiplicative notation. This element g is called a **generator** of the group. A cyclic group is always abelian and may be finite of infinite.

3.2 Rings

In mathematics, a ring F is one of the fundamental algebraic structures used in abstract algebra. It consists of a set equipped with two binary operations that generalize the arithmetic operations of addition and multiplication. A ring must obey to the following rules :

1. $\forall a, b \in F, a + b \in F$
2. $\forall a, b, c \in F, (a + b) + c = a + (b + c)$
3. $\forall a, b \in F, a + b = b + a$
4. $\exists 0 \in F, \forall a \in F, a + 0 = a$
5. $\forall a \in F, \exists - a \in F, a + (-a) = 0$
6. $\forall a, b \in F, a \cdot b \in F$

7. $\forall a, b, c \in F, (a \cdot b) \cdot c = a \cdot (b \cdot c)$

8. $\forall a, b \in F, a \cdot b = b \cdot a$

9. $\exists 1 \in F, \forall a \in F, a \cdot 1 = a$

10. $\forall a, b, c \in F, a \cdot (b + c) = a \cdot b + a \cdot c$

3.3 Fields

In mathematics, a field F is a set with two binary operations + (addition) and $\cdot$ (multiplication) if satisfy all the ring's property plus the following rule :

$\forall a \neq 0 \in F, \exists a^{-1} \in F, a \cdot a^{-1} = 1$. A field $(F, +, \cdot)$ is called a **finite** field if the set F is finite.

In other words, a field is a commutative ring with identity where each non-zero element has a multiplicative inverse.

3.3.1 Galois Fields

In mathematics, a Galois field is a field that contains a finite number of elements.

Theorem For every prime power p^k (k = 1, 2, ...) there is a **unique** finite field containing p^k elements. These fields are denoted by **GF(p^k)**.

Polynomials Polynomial equations and factorization in finite fields can be different than over the rationals. Lets see some examples :

$$x^6 - 1 \text{ over the rationals is} \rightarrow (x-1)(x+1)(x^2+x+1)(x^2-x+1)$$
$$x^6 - 1 \text{ over } \mathbb{Z}_7 \text{ is} \rightarrow (x+1)(x+3)(x+2)(4+x)(x+5)(x+6)$$

A polynomial is **irreducible** in $GF(p)$ if it does not factor over $GF(p)$. Otherwise it is **reducible**.

Theorem Let f(x) be an irreducible polynomial of degree k over Z_p. The finite field GF(p^k) can be realized as the set of degree k-1 polynomials over Z_p, with addition and multiplication done modulo f(x).

3.4 Congruence

As a congruence relation, mod expresses that two arguments have the same remainder with respect to a given modulus. For example, $7 \equiv 4 \bmod 3$ expresses the fact that both 7 and 4 have a remainder of 1 when divided by 3. The following two expressions are equivalent :

$$a \equiv b \bmod m \Leftrightarrow a \bmod m = b \bmod m.$$

Another way of expressing it is to say that the expression $a \equiv b \bmod m$ is the same as saying that $a-b$ is an integral multiple of m. The congruence relation is reflexive, symmetric and transitive, hence it's an equivalence relation. His main properties are :

- **Invariance over addition** : $a \equiv b \bmod n \Leftrightarrow (a+c) \equiv (b+c) \bmod n, \forall a, b, c \in \mathbb{N}, \forall n \in \mathbb{N}_0$
- **Invariance over multiplication** : $a \equiv b \bmod n \Rightarrow a \cdot c \equiv b \cdot c \bmod n, \forall a, b, c \in \mathbb{N}, \forall n \in \mathbb{N}_0$
- **Invariance over exponentiation** : $a \equiv b \bmod n \Rightarrow a^k \equiv b^k \bmod n, \forall a, b \in \mathbb{N}, \forall k \in \mathbb{N}, \forall n \in \mathbb{N}_0$

3.5 Order of Elements

Let a an element of group G. We say that a is of order n if $a^n = 0$ ($a^n = 1$ for multiplicative operator) and for any $m < n, a^m \neq 0$.

Euler totient function In number theory, Euler's totient function count the positive integers up to a given integer n that are relatively prime to n. In other words, it's the number of integers k in the range $1 \leq k \leq n$ for which the greatest common divisor gcd(n, k) is equal to 1, i.e.

$$\phi(n) = |\mathbb{Z}_n^*|.$$

Euler theorem This states that if a and n are relatively prime then

$$a^{\phi(n)} \equiv 1 \bmod n.$$

The special case where n is prime is known as **Fermat's little theorem**.

4 Data integrity and authentication

Our primary goal when we want to exchange some message is to ensure its integrity, even in presence of an active adversary who sends own messages. We want also to be sure that the received message comes from the intended sender and not from the adversary. In other words, we want to enforce also the authentication measure. In order to do this, we can use a technique, which involve the use of a secret key to generate a small fixed-size block of data, known as a **cryptographic checksum** or **Message Authentication Code (MAC)** or tag, that is appended to the message. This technique assumes that two communicating parties, say A and B, share a common secret key K. When A has a message to send to B, it calculates the MAC as a function of the message and the key :

$$C = MAC(K, M),$$

where M represents the input message, K is the shared secret key, MAC is the MAC function and C is the message authentication code. The message plus MAC are transmitted to the intended destination. The destination performs the same calculation on the received message, using the same secret key, to generate a new MAC. The received MAC is compared to the calculated MAC. If we assume that only the receiver and the sender know the identity of the secret key and if the received MAC matches the calculated MAC, then : the receiver can be sure that the message has not been altered and that it comes from the alleged sender. A MAC function is similar to encryption. One difference is that the MAC algorithm need not be reversible, as it must be for decryption. In general, the MAC function is a *many-to-one* function. The domain of the function consists of messages of some arbitrary length, whereas the range consists of all possible MACs and all possible keys. If a n-bit MAC is used, then there are 2^n possible MACs, whereas there are N possible messages with $N >> 2^n$. It turn out that, because of the mathematical properties of the authentication function, it's less vulnerable to being broken than encryption. In fact, the MAC function should satisfy the following requirements :

- if an opponent observes M and $MAC(K, M)$, it should be computationally infeasible for the opponent to construct a message $M^{'}$ such that $MAC(K, M) = MAC(K, M^{'})$.

- $MAC(K, M)$ should be uniformly distributed in the sense that for randomly chosen messages, M and $M^{'}$, the probability that $MAC(K, M) = MAC(K, M^{'})$ is 2^{-n}, where n is the number of bits in the tag.

- let $M^{'}$ be equal to some known transformation on M. That is, $M^{'} = f(M)$. Then $Pr[MAC(K, M) = MAC(K, M^{'})] = 2^{-n}$.

If the adversary would be able to manage to send a meaningless message with the correct authentication tag, this would be an important success for the adversary, because he will trick B to think that he got a message A, but he can't understand the meaning. It's an important success for the adversary, because even if the message is meaningless he's running what is called an **existential forgery**. In the case the adversary is able to choose the message and then construct the correct authentication tag, he get the ability to make what is called the **universal forgery**. Lets see MAC used in CBC mode. Suppose we have a secret key that Alice and Bob are sharing, use this key for making an encryption of the message. We choose our favorite encryptor and we just use the message as the input and we get the ciphertext. Now, we consider the last block of the ciphertext and we say that this is our authentication tag. It means that Alice is running the encryption at her side, uses CBC and gets many blocks, but just discards all blocks except the last one. Doing this job Alice will send the message as a plaintext and the authentication tag. Bob using CBC get the last ciphertext block and compare what he got with what he received from Alice. This approach is secure if the encryption based on key k is looking like a pseudorandom function, satisfying the properties of random functions and if the message have a fixed length. CBC is not secure when we have messages with a variable length. Suppose that the attacker knows two pairs (m, t) and $(m^{'}, t^{'})$, obtained eavesdropping for knowing legal pairs. Starting from this, the adversary can generate a new message, longer than the previous one and the authentication of this new message $m^{''}$ will be again $t^{'}$ (this is an existential forgery). This new message is obtained by concatenating the original message m with a modification of $m^{'}$, obtained by XORing its first block with t. What if we want both confidentiality and data integrity ? We can take the message, composed by some number of blocks, construct CBC using some secret key K_1 and we get the authentication tag. Now, in order to get the confidentiality we construct new ciphertext blocks using another key, the one for confidentiality. So, in this way Alice can send not the plaintext but the ciphertext obtained using K_2 and the authentication tag constructed by taking as input the original plaintext using another key K_1. When we sending the message, we will send all the blocks of ciphertext and send again the last block as authentication tag. This means that whatever adversary can write a ciphertext and just repeat the last block as authentication tag. So, we want two different keys in order to have both properties.

4.1 Unkeyed hashing functions

For now we stop considering the presence of the key, no secret key shared, and lets see what we can do without a key. A hashing function is a function used in several setting in computer science (for example hashing tables). Now, we want to reuse the same concept, the hashing functions is used for computing some properties, we use them to take as input a whole input and get the result, the authentication code. According to the original definition, an hashing function is a function that maps a domain into a smaller domain called **codomain**. We need that for several reasons, for example we need to have short authentication tag to be easily managed. A consequences of the definition we have that the function cannot be one-to-one, it will be not injective. It means that we cannot invert the function, for any possible value in the range we may have several values in the in the domain. When we get this we have what is called a **collision** (a potential danger). Imagine that Alice is sending the message to Bob using two different transmission. First is sending the hash of the file, then she will send the file. Assume that the attacker has not been able to intercept the sending of the hash, Bob gets the hash. Now, a possibility to have success for the attacker, is to have a message with the same hash, same authentication tag. It means there is a collision between the message wanted by the attacker and the original message. A collision can be exploited by the adversary without changing the authentication tag.

Weakly collision resistant A hash function $h : D \to R$ is called **weakly collision resistant** for $x \in D$ if it is hard to find $x^{'} \neq x$ such that $h(x^{'}) = h(x)$.

Strongly collision resistant A hash function $h : D \to R$ is called **strongly collision resistant** if it is hard to find $x, x^{'}$ such that $x^{'} \neq x$ but $h(x) = h(x^{'})$.

Lemma 1. *Strongly collision resistant implies weakly collision resistant.*

Proof. To prove this claim we show that $\neg$ weakly $\implies$ $\neg$ strongly. Given h, suppose there is a polynomial algorithm $A_h : A_h(x) = x^{'}$ such that $h(x) = h(x^{'})$. We construct another polynomial algorithm $B_h : B_h() = (x, x^{'})$ such that $h(x) = h(x^{'})$ (where x is randomly choose, and $x^{'} = A_h(x)$. □

4.2 Keyed hashing functions

Now we want to add a secret key to a hash function for example to authenticate two parties in a communication. Hash function has been designed to take as input the message, they don't show the possibility to get two inputs, if you want also a key you have to mix the message and the key, you can do that is so many approaches, like :

- suppose we want to compute the authentication tag in a message by using a secret key and in order to do that we use the M-D construction and we compute the digest of a string obtained by concatenating key and message. It happens that the first block depends on the length of the key. Next blocks will just depend on these message blocks. So, suppose that the first block is just the key, you get the digest of the message and this is depending on the key. If the adversary can compute the authentication tag, without knowing the key, of the same message adding one extra h block, and then the two inputs, one input is the result of this block, the other is next block. So this approach is weak because if we use the key as a prefix of the content it's easy to prolong the message using whatever the adversary wants, adding whatever message block and using this output.

- suppose we want to use the same previous construction, but concatenating the message and then the secret key. The problem here comes if it happens that the key at the end this time appears to be the last block. Keys are not so big in this case but may happen that the size of the key is just the size of the last block. The last block is taking the key as input. In this case, if the adversary is able to find a collision by using the hashing functions between two messages, he can replace the original message with a colliding message having the same size. He doesn't need to know the key, he can take an original pair, can replace the message, with the colliding message found by the birthday attack.

- a good approach is prepending and appending the key to the message. The first method is useless since the adversary cannot just add extra block, he need to known the key because the last block is a key. In the second case, is not sufficient for the adversary to find a collision message and he's not knowing the key. If the adversary doesn't known the key, can try to generate messages but can't recognize when he generates a good collision.

In the third case we can still use the birthday attack ? It's still works, but is

useless. In fact, after that the adversary finds a collision, let's say $x^{'}$ and $x^{''}$, he can manage by using the birthday paradox, so not too many attempts. The adversary doesn't have a possibility to known if this collision have the same prefix and the same suffix. That is the real problem for this attack in this situation; the attacker can't verify its success.
Now, suppose there is an oracle, this one is computing the HMAC using some fixed key (unknown to the attacker), for any message you can ask. When you will ask you query to the oracle you provide the message, then the oracle will compute the HMAC of the message by using some fixed key. So, since the output of HMAC have a fixed number of bits, if such number of bits is n, then you expect that after $2^{\frac{n}{2}}$ different messages you get a collision with probability of 0.5. This is the meaning of the birthday attack. So, calling $m1$ and $m2$ such collision, you can choose a random string x, and ask to the oracle to compute another HMAC over the concatenation of $m1$ and x. Now, this result named t with a very good probability it will the same authentication tag of message $m2$ concatenated with x (for iterated hashing function). So, you found another collision without asking the oracle for another query.

4.3 Birthday paradox

The problem can be stated as follows. What is the minimum value of k such that the probability is greater than 0.5 that at least two people in a group of k people have the same birthday ? Assuming that each birthday is equally likely. The probability that the birthdays of any two people are not alike is clearly $\frac{364}{365}$. The probability that a third person's birthday will differ from the other two is $\frac{363}{365}$, and so on until reach the 24th person. We thus obtain a series of 23 fractions which must be multiplied together to reach the probability that all 24 birthdays are different. The product is a fraction that reduces to about $\frac{1}{2}$. More generally, let $h : D \rightarrow R$ be any mapping. If we choose $1,1774 \cdot ||R||^{\frac{1}{2}}$ elements of D at random, the probability that two of them are mapped to the same image is 0.5. Real world applications for the birthday paradox include a cryptographic attack called the **birthday attack**, which uses this probabilistic model to reduce the complexity of finding a collision for a hash function. In fact, suppose that we have a function f, the goal of the attacker is to find two inputs x_1 and x_2 such that $f(x_1) = f(x_2)$. The attacker can proceed in a brute-force manner, and thanks to the birthday attack he expects to find a collision after $1.1774 \cdot ||R||^{\frac{1}{2}}$, where R is the codomain of the function f. This value is called **birthday bound**.

4.4 Cryptographic hashing function

We accept cryptographic hashing functions having some properties, in particular we want hashing functions that are strongly collision resistant. Another good property is that the hashing function must be fast to be computed. A cryptographic hash function should resist to attacks on its preimage (set of possible inputs). In the context of attack, there are two types of preimage resistance :

- **Preimage resistance** : for essentially all pre-specified outputs, it is computationally infeasible to find any input that hashes to that output, i.e. given y, it is difficult to find an x such that $h(x) = y$.
- **Second-preimage resistance** : it is computationally infeasible to find any second input which has the same output as that of a specified input, i.e. given x, it is difficult to find a second preimage $x^{'} \neq x$ such that $h(x) = h(x^{'})$.

What we need is not the original hashing function working on a single block, we need the result of some construction useful to make the function working for a document whatever is its length (**Merkle-Damgard construction**). It's a construction for building collision-resistant cryptographic hash functions from collision resistant one-way compression functions. Its hash function first applies an MD-compliant padding function to create an input whose size is a multiple of a fixed number. The hash function then breaks the result into blocks of fixed size, and processes them one at a time with the compression function, each time combining a block of the input with the output of the previous round. In order to make the construction secure the messages must be padded with a padding that encodes the length of the the original message. This is called **length padding** or **Merkle-Damgard strengthening**. If the basic function H is collision resistant, then so is its extension. Finally, the output length should be at least 160 bits to prevent birthday attacks.

4.5 SHA-1

To-do

4.6 HMAC

In recent years, has been increased interest in developing a MAC derived from a cryptographic hash function. The motivations for this interest are that cryptographic hash functions generally execute faster in software than symmetric block ciphers and there is a widely available library code for them. It receives as input a message m, a key k and a hash function h, and simply output a MAC in the following way :

$$HMAC_k(m,h) = h(k \oplus opad||h(k \oplus ipad||m)),$$

where opad ($5c$ in hexadecimal) and ipad (36 in hexadecimal) are the outer and inner padding respectively. HMAC can be forged if and only if the underlying hash function is broken. Furthermore, with HMAC the birthday attack becomes useless, even if the attacker can find a collision, he can't in any way check the success without knowing the key (even in the case of $h(k||m||k)$).

4.7 Authenticated Encryption

It is a term used to describe encryption systems that simultaneously protect confidentiality and integrity of communications. There are several approaches to do so :

- **Encrypt-then-MAC (EtM)** : this approach use two keys. First encrypt the message to obtain the ciphertext. Then authenticate the ciphertext with a MAC using a key different from the previous one.
- **Encrypt-and-MAC (E&M)** : it uses two keys. Encrypt the message to obtain the ciphertext. Authenticate the plaintext with another key. These operation can be performed in either order.
- **MAC-then-Encrypt (MtE)** : it uses two keys. First authenticate the plaintext by computing the MAC value. Then encrypt the message plus the tag using a different key.

Authenticated Encryption with Associated Data is a variant of Authenticated Encryption that allows a recipient to check of both the encrypted and decrypted information in a message. It binds associated data to the ciphertext and to the context where it's supposed to appear so that attempts to "cut-and-paste" a valid ciphertext into a different context are detected and rejected.

5 Public-Key Cryptography

Public-key cryptography is just one type of asymmetric cryptography. If there is a group of n people that wants to communicate, then each pair of people has a different private key to use, for a total of n^2 keys to be stored. We can enhance this approach by looking to the **Diffie-Hellman** approach.

5.1 Diffie-Hellman

The purpose of Diffie-Hellman approach is to enable two users to securely exchange a key that can then be used for subsequent encryption of messages. In the previous scenario, instead of generating one key for each pair of people, everyone has its own pair, i.e. we need to store only $2n$ keys. The original key is divided in 2 parts : **K_E** to be used for encrypting messages and **K_D** to be used for decrypting messages. The first key is public, while the second one is private and must be kept secret. Before continuing our discussion on Diffie-Hellman approach we need to introduce the concept of :

One-way function A function $f : \{0,1\}^* \rightarrow \{0,1\}^*$ is one-way if f can be computed by a polynomial time algorithm, but any polynomial time randomized algorithm F that attempts to compute a pseudo-inverse for f succeeds with a negligible probability. That is, for all randomized algorithms F, all positive integers c and all sufficiently large $n = length(x)$, we have that

$$Pr[f(F(f(x))) = f(x)] < n^{-c},$$

where the probability is over the choice of x from the discrete uniform distribution on $\{0,1\}^n$, and the randomness of F. By this definition, the function must be "hard to invert" in the average case, rather than the worst case sense. However, the existence of one-way functions is still an open problem. If a one-way function exists then $P \neq NP$, but it's not known the viceversa. The Diffie-Hellman approach is based on the idea that the product $N = p \cdot q$ of 2 integers is easy to compute but the factorization is hard, which is supposed to be a one-way function. In other words, it's based on the difficulty of computing discrete logarithms.

Discrete logarithm Let G be a finite cyclic group with n elements and g a generator of G. Let y be any element in G, that it can written as $y = g^x$, for some integer x. Let $y = g^x$ and x the minimal non negative integer satisfying the equation. The minimal x such that $y = g^x$ is called the **discrete log** of y to base g. So, the discrete log is believed to be a one-way function, because :

- $x \rightarrow g^x \bmod p$ is easy to compute

- $g^x \bmod p \to x$ is hard to compute.

Now lets see in details the Diffie-Hellman key exchange algorithm. The idea is the following, if we let Alice and Bob talk in front of the public and they exchange messages, the public of course is listening, and Alice and Bob determine a secret key but the public is not able to take this information, why ? We want Alice and Bob that can choose a number that will be a secret key and we want they can do this without the knowledge of any shared information, they can determine this secret key by public messages. Alice and Bob and the public know the algorithm used, no secret at all. Actually the two parties don't need to exchange many messages (two messages are sufficient). Alice choosing some numbers p, g and a. Then, she computes $g^a \bmod p$ and sends the result and numbers g and p to Bob. Bob receives this message, choose a number b to compute $g^b \bmod p$ and sends the result back to Alice. Now, Bob and Alice can compute the shared key $g^{ab} \bmod p$ because :

- Bob holds b and computes $(g^a)^b \bmod p = g^{ab} \bmod p$.
- Alice holds a and computes $(g^b)^a \bmod p = g^{ab} \bmod p$.

What is somewhat surprising is that the result of such computation is the same number K. The number chosen by Alice and Bob are not public, because they are needed to compute K. Now, we can see the Diffie-Hellman key exchange. The public information are a prime p, an element g possibly a generator of the multiplicative group $\mathbb{Z}_p^*$. In this way, the procedure can continue by letting Alice choosing a number a at random, inside the interval $[1, p-1]$. It's strong because is strong the discrete log. Since the integers a and b are discarded at the end of the session, DH is offering what is called the **perfect forward secrecy**. It means that, if for some reasons an attacker manages to find the key, he will get no information about the previous key and the future keys to be established. Another fundamental thing to say, is that the key exchange by DH is weak against a **MITM** or **bucket-brigade** attack. In this case, the attacker could send a message to Bob claiming to be Alice. When the attacker send this message, he will forwards g and p, but he will not send $g^a \bmod p$ since he will choose another number. He will be able to fix a shared secret key between Alice and man in the middle. Allowing the man in the middle to setup another key shared between itself and Bob. If we are under a man in the middle attack, the attacker can get success of fixing a secret key between himself and the two parties. The two parties will be unaware of that, instead they fixed a secret key with the man in the middle. To fix this problem we need an authentication process between the two parties. DH approach is weak against the **logjam attack**. In fact, it allows a MITM attacker to use an old version of the protocol. The old versions of DH always use the same g and p, but these could be precomputed in reasonable time. Now the question to ask ourselves is the following : Is the one-way relationship between public information and shared private key sufficient ? A one-way function may leak some bits of its arguments. We can prevent this, carefully choosing g and p.

5.2 RSA

Our goal is to encrypt by the public key and decrypt by the private key or viceversa. Of course the algorithm for implementing this idea was not found by DH, it was found one or two years later. In particular, the researchers are trying to focus on the idea of using one way function to be easy to compute in one direction and hard to be done in the other direction. In order to work on this idea, is nice to consider the multiplicative group Z^*_{pq}. This is defined as usual multiplicative group, where p and q are two prime numbers (possibly large). Let N be the product of these two number. Of course N is not prime, since it has two factors. The content of Z^*_N are all numbers co-prime to N (i.e. co-prime to q and to p). Now, we want to know, how many elements are in this set ? The name of that cardinality is **Totient function**, in our case $\phi(N)$, introduced by Euler. All members of Z^*_N should belong to the interval $[1, p \cdot q - 1]$. Since in this interval we have $(p-1)$ multiples of q and $(q-1)$ multiples of p, the size of this group is $\phi(pq) = pq - 1 - (p-1) - (q-1) = pq - (p+q) + 1 = (p-1)(q-1)$. Now, we want to exponentiate for encryption, and for this reason we pick a number $e \in (1, (p-1)(q-1))$. We are interested to determine which values of e makes $x \rightarrow x^e$ a one-to-one mapping. If e is relatively prime to $(p-1)(q-1)$ then $x \rightarrow x^e$ is a one-to-one mapping in Z^*_{pq}.

Proof. Since $gcd(e, (p-1)(q-1)) = 1, e$ has a multiplicative inverse, denoted as d, $\bmod (p-1)(q-1)$. Then $e \cdot d = 1 + C(p-1)(q-1)$, for some constant C. Let $y = x^e$, then $y^d = (x^e)^d = x^{1+C(p-1)(q-1)} = x^1 \cdot x^{C(p-1)(q-1)} = x(x^{(p-1)(q-1)})^C = x \cdot 1 = x$, meaning that $y \rightarrow y^d$ is the inverse of $x \rightarrow x^e$. □

So, let $N = pq$ be the product of two primes numbers. Choose $1 < e < \phi(N)$ such that $gcd(e, \phi(N)) = 1$. Let d be such that $de \equiv 1 \bmod \phi(N)$. Then the public key is (e, N) and the private key is (d, N). For encrypt a message M we will compute $M^e \bmod N$, while for decrypting a ciphertext C we will compute $C^d \bmod N$. Why the decryption works ? We have found that by definition d is the multiplicative inverse of e mod $\phi(N)$. This means that there exists an integer k such that e times d is equal to $1 + k \cdot \phi(N)$. Consider m and p (m is the message, the number associated to the message and p is a prime number we have chosen to run RSA). Since p is a prime number, either m is co-prime to p, or m is a multiple of p. In the first case, $gcd(m, p) = 1$, by the Fermat theorem that is, in this case, equivalent to the Euler theorem, m^{p-1} is congruent to 1 mod p. Starting from this congruence, we can raise both sides to the same power, $k(q - 1)$ and then multiply both sides by m. We get $m^{1+k(p-1)(q-1)} = m \bmod p$. Such congruence still holds in this case. If it happens that the $gcd(m, p) = p$, you have that both sides are congruent to $0 \bmod p$, and we can write $m = jp$. The exponent of the first side is equal to ed. It happens that m^{ed} is congruent to m mod p. The same reasoning can be done comparing m with another prime number q, getting that m^{ed} is congruent to m mod q. Since p and q are two prime numbers, they are co-prime of course, and it happens that if we are considering the module computing by multiplying p and q, we can combine together these congruences to get the more general congruence m^{ed} mod N. So we are using the property that e and d are the multiplicative inverse of each other. By stating these properties and exploiting the fact that they are co-prime, we can get the combined congruence, this is the process.

5.2.1 RSA algorithm

This allow us to build the RSA algorithm. It works with different key lengths, of course the longer is the key the higher is the security (512 bit is a common choice). We want that the number of encoding the plaintext isn't greater than N, in order to avoid a lot of weak cases. The algorithm is not so fast, since we have to run the exponentiation having the exponent very big. Typically, RSA is significantly slower than DES (public key encryptors slower than symmetric key encryptors). Now, the question is how can we compute the multiplicative inverse d ? We can leverage the following mathematical result :

Bezout's identity $au + bv = d$ ($a \geq b$; u, v signed integers; $d = gcd(a, b)$ + extended Euclid's algorithm). If a and b are co-prime then $d = 1$, u is the multiplicative inverse of $a \bmod b$ and v is the multiplicative inverse of $b \bmod a$. Remembering the traditional Euclid's algorithm, for computing $gcd(x, y)$ calculates $r_n = r_{n-2} \ \% \ r_{n-1}$, with $r_{-2} = x$ and $r_{-1} = y$. When $r_n = 0$ gcd has been found, equals to r_{n-1}. (for the proof see the slides).

5.2.2 RSA instance

Alice can start constructing an instance of RSA by taking two large prime numbers, p and q, and let N be their product (easy task). Then Alice chooses at random a number co-prime to $\phi(N)$, and compute the multiplicative inverse of such number mod$\phi(N)$. This makes possible to define a public and private key. Alice must keep secret these numbers, because if an adversary has the knowledge of them, then he can compute the multiplicative inverse and figure out the private key. The important thing is that when we encrypt we use N, and given N to the adversary is very hard to find p and q (factorization). Now, lets talk about possible implementation of RSA following these steps :

- **Find two random primes** : it's not so easy to accomplish this task. We will generate a huge number of bits, where the last bit is one (odd integer). Then test if the integer is a prime number. In negative case, restart the process. This is a probabilistic algorithm based on a strong theorem called the **prime number theorem**. If you consider this interval from n to 2n inside it there are many prime numbers, around $n \cdot \ln(n)$ (expected to find a prime number every $\ln(n)$ attempts).

- **Encoding** : we have to encode our message that is different from encrypting. The first one means using a code to represent an information, while the second means compose the ciphertext. We can compute exponentiation in some fast way, requiring a linear number of operations ($\log(N)$ size of the input).

We would like to make more clear the idea of one way function. Starting from x, computing $x^e \bmod N$ is easy. We think that is hard to compute x from this information. The **one-way trapdoor function** has been defined. It's a function behaving like a one way function, very hard to be inverted, but if we provides some extra information then the computation becomes easy. If we want to use a one way function it needs to be a one way function with a trapdoor.

5.2.3 Attacks on RSA

Now we want to discuss some RSA attacks in order to specify some its implementation details so that all its weaknesses are removed. First of all we want to introduce the following theorem :

Chinese Remainder theorem : Suppose $n_1, n_2, ..., n_k$ are positive integers which are pairwise co-prime. Then for any given integers $a_1, a_2, ..., a_k$, there exists an integer x solving the system of simultaneous congruences, i.e. $x \equiv a_i \bmod n_i$, for $i = 1, ..., k$. Furthermore all solutions x to this system are congruent modulo the product $N = n_1 n_2 .... n_k$.

There are several attacks that can be performed to break RSA such as :

- **Factor N** : we can find$p$ and q given N. This is believed to be hard unless p and q have some bad properties. We should pick these numbers not too close together, must be large enough (at least 100 digits each) and make sure that both $(p-1)$ and $(q-1)$ have large prime factors.

- **Simple messages** : some messages may be easy to decrypt. If you just apply RSA according to the definition you can get some ciphertext that is easy to decrypt, they are just particular cases. For example if the message $m = 0$, then the ciphertext will be equal to the plaintext (same case for number one). This strange property occurs in the case where the message is equal to $0, 1$ or $N - 1$. First, these is not so frequent that we are encrypting these numbers. We use an approach based on **salting**, adding extra bits using some approach, also random bits. If both m and e are small (e.g. $e = 3$), it happens that m^e is smaller than N. Computing the root of this number is very easy in arithmetic. In this case the solution is adding some extra bits to the original number to make it big enough. If we choose e to be a small number (like three), and if happens that there are two similar messages, m and $m + 1$, then encrypting them we got two different ciphertext c_1 and c_2, and we can easily write a system equation fully defining such information that is allowing to extract the plaintext message m by simple computation. The solution in this case is to choose a large e.

- Assume the exponent is small (e.g. $e = 3$) and that we are sending the same message m three times to three different users. We can expect that different users will have different numbers N, they are choosing different p and q. So the same message is obtained by computing $m^3 \bmod n_1; \text{mod} n_2; \text{mod} n_3$. By using the Chinese remainder theorem you can compute $m^3\underline{m}odn_1n_2n_3$. There is a very high probability that the numbers are co-prime, each of them is the product of two large primes. The message m is smaller than n_1, n_2 and n_3. As a consequence m^3 will be smaller than the product of the three integers. So, we have that $m^3 \bmod n_1 n_2 n_3 = m^3$, and for the attacker its easy to compute the cubic root and get the original message. The solution is to add random bytes to the message in order to avoid equal messages.

- Another vulnerability is related to the fact that the message space is small, it means that even if your message is composed by many bits may be the message should respect some pattern, just because the message is a word belonging to some language. If the legal words of the language are few, then the attacker can compute and encrypt all of them using the public key. Its easy for the attacker to check if the ciphertext is one of that computed. The solution is to add random string to the message.

- Other possible vulnerability is related to the fact that two different users have the same number N, but different e and d. So, if it happens, the owner of the private key can write a set of equations, since he knows both e and d, and he can manage to find p and q without factoring number N, but exploiting the knowledge of N. Such person gets p and q, and if some other person have the same number N, then the first person is able to attack the second person, in order to figure out the private key starting from the public key. The solution of this issue is let people choose by themselves number N, so that the probability that two people share the same number N is very very low.

- One of the most famous properties of RSA is the multiplicative property; if we have a message m, and this number happens to be the product of two numbers m_1 and m_2, when we encrypt $m_1 \times m_2$, computing $(m_1 \cdot m_2)^e \bmod N$, the result will be the product of the encryption of m_1 and the encryption of m_2 mod N, i.e. $(m_1^e \bmod N) \cdot (m_2^e \bmod N)$. So, an adversary can proceed using small messages. This attack belongs to the family of the chosen ciphertext attack. Suppose the adversary want to decrypt some ciphertext C, and he knows that this ciphertext has been obtained by some unknown message m raised to the power of e mod N. The adversary can very easily compute a new number X given by $(C \cdot 2^e) \bmod N$. If the adversary computes such a number, he can run the chosen ciphertext attack and can ask the oracle for decrypting such message while once it has been encrypted, so in practice he can ask the oracle to obtain number $Y = X^d \bmod N$.

According to what we know about the original message X, we can write $X = (C \bmod N) \cdot (2^e \bmod N) = (M^e \bmod N) \cdot (2^e \bmod N) = (2M)^e \bmod N$. So, while decrypting X the adversary will get the knowledge of $2M$. If the adversary manage to find the decryptor free he can use that asking to the oracle to make the decryption of some ciphertext. The typical solution against this type of attack is requesting that the message should respect some structure, it means that the oracle doesn't answer if the message doesn't verify the requirements.

- If the adversary is able to analyze your implementation, he can try to analyze the way your implementation is working by checking the time required to compute some values, or just checking how much energy has been used for computing such an encryption. Decryption is much more difficult. The solutions is adding some random steps.

We notice that the classical textbook implementation of RSA is not safe. Several attacks are possible in many cases. The idea is fine, we want to encrypt a message m, preprocessing it we got a new message $m^{'}$ and then we apply RSA to the new message. Of course we need that these two messages have the same meaning, otherwise we can have a very easy function that gives us $m^{'}$ starting from m and viceversa.

5.3 Public-key Cryptography Standard

The result of the research about good implementation of RSA is known as **public key infrastructure**. An infrastructure, because we need a lot of standards defining how to do the steps for a good implementation of RSA. One of the first result proposed by the RSA company and immediately standardized as internet standard, was called **public key cryptography standards (PKCS)**. It's interesting to see the first standard (PKCS#1), which define a standard to send messages using RSA. If we start from the original message M, we are going to encrypt another message m, which start by zero (it implies that message is less than N), concatenated with a code saying that we want to do encryption (number two), some random bits, a zero and then the message M. In this way each time we do an encryption we get a different real number, providing a good property since the attacker can't even check if Alice is sending the same message to Bob. However, today it's obsolete. Today there are stronger approaches for preprocessing a message m such as **Optimal Asymmetric Encryption Padding (OAEP)**. This is a way of padding the original message for contrasting all the types of attack we have seen and it's also providing a fast way for encoding and decoding the message. The idea is take the message, preprocess it obtaining a new message, and then encrypt it. Let n be the number of bits, k_0 and k_1 are integers that are fixed by the implementation, and our message m is constituted by $n - k_0 - k_1$ bits. Then we just pad this message with a sequence of k_1 zeroes. We have another string r, that is a random string with length k_0. In this schema we are going to use two functions, G and H. Typically, they are just hashing functions. It happens that we use the string r as input for G, the output of the function is XORed to the padded message giving raise to the part X of the message to be encrypted. In order to obtain the part Y we provide X as input to H and then the result is XORed with r. This is a way in which we encode the original message, we get an encoded message, this is the message really encrypted. The receiver will run the decryption algorithm obtaining X and Y and recover the original message very easily. This proposal is a very strong result, making thing for developers much more complicated, not because the schema is complicated, but if we start checking libraries offering encryption and decryption function when we begin with our crypto-graphical security we won't write the libraries again, we will choose good libraries. The security provided by this scheme is called **All-or-nothing security**. It means that in order to find the original message m, we must recover the whole messages X and Y. So by just modifying one bit of X and Y we get the result of destroying the possibility to completely recover m, since we have the role of the hashing functions. You need to have all bits correct or you get nothing.

5.4 ElGamal Encryption

What ElGamal was able to do is find a solution to DH approach using the same mathematical approach. It's constituted by three components :

1. **Key generator** : first of all Alice publishes $p, g \in Z_p$, where g is the generator of a cyclic group of order p. Then Alice chooses x as a private key and publishes $g^x \bmod p$ as a public key, where $x \in \{0, 1, ..., p-1\}$ chosen at random.

2. **Encryption algorithm** : Bob in order to encrypt a message $m \in Z_p$ for sending it to Alice, send to her $g^y \bmod p$ and $m \cdot g^{xy} \bmod p$, where $y \in \{0, 1, ..., p-1\}$ chosen at random.

3. **Decryption algorithm** : Alice computes $(g^y)^x \bmod p = g^{xy} \bmod p$, and then computes $(g^{xy})^{-1} \bmod p$ for obtaining $m \cdot g^{xy} \cdot (g^{xy})^{-1} \bmod p = m$.

The validity of this approach is limited by all this math and computation effort (in fact it requires two exponentiations for each block transmitted). However a new y is chosen for every message to transmit in order to improve security.

6 Digital signatures

Digital signatures are used for guarantee data integrity and non repudiation. We studied hashing functions, we can use them for computing the tag and use keyed hashing functions, so that Alice and Bob will be guaranteed that it's the counter part sending the files. In a perfect world everything is ok, but the world isn't perfect. If Alice says it was Bob by sending that file, and Bob replies not, it has been Alice sending the authentication tag for that file. For sending it, is required the knowledge of the key, but both Alice and Bob may know the key. How to solve this problem ? We want to be sure about the originator of the message. This is a limit of the approach based on MAC. Lets see a wrong candidate for the digital signature scheme. Suppose that Bob wants to send a message M to Alice. It will send the pair $(M, E_{K_{PVT}})$ to Alice, where K_{PVT} is Bob's private key and $E_{K_{PVT}}$ represents the digital signature of the message M. Now, Alice can verify the integrity and the non repudiation property give in input to the algorithm M, the digital signature and Bob's public key. Now, if an attacker changes even one bit of the message M, the validation will fail. Furthermore, the pure DH approach is subject to the **existential forgery** attack. In this case the attacker is able to find a pair, like message (even meaningless) and signature, such that who is verifying the signature will obtain accept. Because if we send to Bob a random sequence of bits and a digital signature and Bob makes the verification and he says ok accept, Bob will say Alice was the sender of this message. Since the security of digital signature is very high for authenticating the sender, Bob will be sure about the originator of the message, so an existential forgery seems a weak attack but it's very bad. There are two other types of forgery attack. The **selective forgery** is an attack where the adversary have the ability of creating a pair constituted by message and signature, where the message is chosen in advance before running the attack satisfying some mathematical property with respect to the signature algorithm. If he can make a selective forgery attack, he is also able to make an existential forgery. The strongest attack is called **universal forgery**, in which the adversary has the ability to decide whatever message and to show the correct signature for that message. This attack implies the other types of forgery. If we would like to develop some solution to be strong against these types of attack, we will focus on existential forgery since $\neg existential \implies \neg selective \implies \neg universal$. With the RSA approach, since it's multiplicative, than given two message M_1 and M_2 and a private key D_A then we will have $D_A(M_1M_2) = D_A(M_1)D_A(M_2)$. To avoid this security issue, indicating with D_A Alice's private key and E_A Alice's public key, we should first compute the hash of the message M, i.e. $y = H(M)$, then encrypt it obtaining $z = D_A(y)$ and send (M, z). On the destination side we will have to decrypt the signature, i.e. $y = E_A(z)$ and check if $y = H(M)$. Naturally, the hash function H should be collision resistant, so that an attacker cannot find another $M^{'}$ such that $H(M) = H(M^{'})$.

6.1 Signature scheme

The signature scheme is very similar to the one used for RSA encryption, which is constituted by the following building blocks :

- **Generation** : this step involves the generation of private and public key in a random way.
- **Signing** : we encrypt the hash, also called digest, of the message using the private key. This phase can be either deterministic or randomized.
- **Verification** : on the destination side the verification process is performed in order to accept or reject the message. Usually, this phase is deterministic.

6.1.1 RSA

Using RSA the signature is represent by the encryption, using the private key, of the message digest. In this case, only the person who knows the private key can sign the message, and everybody can verify the signature using the public key. We can refers to PKCS#1 in order to use RSA for building a digital signature for a given message. Typically, it uses a **Signature Scheme with Appendix (SSA)**, which means that the appendix is the signature and it's added to the message. There exists two approaches that differ in how the encoded message is obtained :

- **RSASSA-PSS** : it's an improvement of the probabilistic signature scheme with appendix. It uses a signature scheme called **EMSA-PSS**, which is an encoding method for signature appendix. It's inspired to OAEP, in fact, given a message compute its hash. Some padding bytes are concatenated with this result and then concatenated with some salt bytes, in order to build a new message $m^{'}$. Then we compute the hash of $m^{'}$, that we denote with H. Then we give it in input to a masked generation function, and the output will be XORed with some bytes of salt. This result is concatenated with the value H.

- **RSASSA-PKCS1-v1_5** : it uses $EMSA - PKCS1 - v1\_5$, but it's used only for compatibility reasons.

6.1.2 ElGamal

We have an approach, still by ElGamal, that defines a signature scheme. First, we need to choose a prime number p with 1024 bits. We want that the discrete logarithm is hard in this group. Let g a generator of Z_p^*. Now choose a random number $x \in [2, p-2]$. Then compute $y = g^x \bmod p$. We assume that the public key is (p, g, y). The private information is the exponent x we chosen before. This process is called **generation**. If we want to sign a message m we define a new pair of keys, because the proposal is lets make a standard usage of our keys. Lets also generate temporary keys for running encryption, for every signature we are generating extra keys. In this way, every time we sign the same document we will get different signatures, but the process of the verification must work in any case. First, we hash the original message with a crypto-graphical quality hashing function. Now, we can pick a number $k \in [1, p-2]$, which is relatively prime to $p-1$ and compute $r = g^k \bmod p$. We notice that r doesn't depends on the message and this implies we can apply some preprocessing technique preparing a table of numbers. It's not really recommended, you will have the problem of putting in a safe way such information, anyway compute this number r, is the result of another exponentiation, and now compute number $s = (m - rx)k^{-1} \bmod (p-1)$. If it happens that we get zero from this computation, try again picking another random number. By ElGamal approach we get the pair (r, s). If we use this type of signature when we will send our message, we will send a pair, the message and the second item of the pair is this pair, i.e. $(m, (r, s))$. The signature is a pair. Now suppose we want to verify this signature. The number r should be less than p and number s should be less than $p-1$ and $y^r r^s = g^m \bmod p$ to accept, otherwise reject. In fact the relation for obtaining s can be rewrite as $ks + rx \equiv m \bmod (p-1)$. We know that $r = g^k \bmod p$, so $r^s = g^{ks} \bmod p$, and $y = g^x \bmod p$, thus $y^r = g^{rx} \bmod p$. The previous relations implies that $y^r r^s = g^m \bmod p$.

6.2 Digital Signature Standard

Its inspired to ElGamal approach and according to the original formulation, it uses SHA because the standard hashing function chosen is SHA family. People are talking about DSS and about DSA, lets see the difference. DSA is just the algorithm to achieve the digital signature, DSS is the standard, i.e. a set of further details. Lets talk about about the algorithm. The idea is working with two prime numbers p and q (160 bit), such that p is a multiple of q plus one and the discrete logarithm mod$p$ is intractable. Once we agreed on these two numbers, we need to focus on another number denoted as α. That particular number is such that if you compute $\alpha^q = 1 \bmod p$ or $\alpha = 1^{\frac{1}{q}} \bmod p$. In order to compute α we need to take a random number $h \in (1, p-1)$, and compute another number $g = h^{\frac{p-1}{q}} \bmod p = h^j \bmod p$, since $p = j \cdot q + 1$, for some j. Now, if $g = 1$ we need to choose a different h because the things would be insecure. Otherwise, it holds that $g^q = h^{p-1}$. Now, by Fermat's theorem we have that $h^{p-1} \equiv 1 \bmod p$. In other words we found a number g such that raised to the power of q is congruent to 1 mod p (it's α). We choose α to be equal to g. Starting from such information we generate private and public key. The private key is a random number $s \in [1, q-1]$. The public key is constituted by $(p, q, \alpha; y = \alpha^s \bmod p)$. By hypothesis is hard to find s since we need to compute the discrete log. The signature of a message M, starts by choosing a random number $k \in [1, q-1]$, that is kept secret. Its a pair formed as follows :

- **Part I** : we compute $P_I = (\alpha^k \bmod p) \bmod q$. It doesn't depends from the message.
- **Part II** : it depends from the message, we get the digest of the message using an hash function. Then we sum it with the product of s and $PartI$, all multiplied by multiplicative inverse of k and then we apply mod$q$, i.e. $P_{II} = (SHA(M) + s(P_I))k^{-1} \bmod q$. It's very fast to compute.

Now, we need to define also the verification scheme. We need to compute two numbers $e_1 = SHA(M)(P_{II})^{-1} \bmod q$ and $e_2 = P_I(P_{II})^{-1}$ mod q. We will accept the signature if $(\alpha^{e1} y^{e2} \bmod p) \bmod q = P_I$. Lets see why this scheme works. From P_{II} definition we get $SHA(M) = (P_{II}k - sP_I) \bmod q$, thus we have $\frac{SHA(M)}{P_{II}} + s\frac{P_I}{P_{II}} = k \bmod q$. The definition of $y = \alpha^s \bmod p$ implies $\alpha^{e1} y^{e2} \bmod p = \alpha^{e1} \alpha^{se2} \bmod p = \alpha^{e1+se2} \bmod p = \alpha^{\frac{SHA(M)}{P_{II}} + s\frac{P_I}{P_{II}} \mod q} \bmod p = \alpha^k \bmod p$. Now, the execution of mod$q$ implies $(\alpha^{e1} y^{e2} \bmod p) \bmod q = (\alpha^k \bmod p) \bmod q = P_I$. The fundamental thing is to protect the number k our temporary private key, otherwise the attacker knowing the public key and k can setup a system and get sensitive information. When the adversary know k, he will able to run new signatures for new messages. The ability of knowing the key is allowing the adversary to sign, the power of universal forgery. The analysis of the mathematical security of DSA is strong enough. DSA is obtained by ElGamal. Not use ElGamal, is good, let's improve ElGamal to obtain a standard, this means that in practice ElGamal is not used. The signature verification step for DSS is more demanding that RSA. For what concerns signing documents, more or less the two approaches have the same speed. Still use preprocessing, we will be a little faster than RSA in making signature. DSS requires to generate random numbers. They are used as private keys, and should be chosen is a good way. In many cases proprocessing is used when we want to implement signatures with smart cards, this makes the process faster and we want to save energy for the process of running the algorithm. DSS is used for just signature, RSA both for signature and key management. DH just for key management. There is another point, when you are signing an analogic contract, they give you a paper and you sign. You need to specify the time. When we have an important signature, two parties are agreeing of something, in many cases is requested the time and we have seen in the process of taking a digital signature that there isn't a mention of time and day. In those cases we need to timestamp a document, to associate in a secure way a timestamp. If we want to associate an official time the typical approach is using a third party providing the service of generating timestamps. Suppose that Alice want to timestamp a document, of course while timestamping Alice will have a secure hash and send it to the authority (named **TimeStamping Authority (TSA)**). It adds the timestamp, it's just a string containing the month, year, day, time, hour, seconds, and computes the hash of the received hash plus the timestamp. After that, it sign the value obtained and send back to Alice. She obtains the statement and the timestamp authority associates a timestamp to the hash. This association is certified by a digital signature.

7 Authentication

Suppose a scenario in which Alice needs to show her identity to Bob because she wants a service, and Bob will give the service only to authorized people. Bob wants to know if Alice is really Alice, if she belongs to the list of people having the right to use the service. But what is the identity ? The digital identity is a very complex concept, we can consider it like our credentials for reading our emails. We are a user of an email system, that's a digital identity. So, Bob wants to be sure about Alice, so Alice can be just a pair, just an email address associated to a real person that has been verified. In this case the attacker can perform two type of attacks, MITM and spoofing attack. These attacks somewhat overlaps because in some cases it's not easy to say they are different, in some cases they are the same attack. For what concerns authentication the word most frequently used is **spoofing**, which means that i cheat about the identity of the originator of some messages, an email message or a datagram, sending packets or whatever, at different levels of the stack ISO/OSI. There are many spoofing attacks, so that the intruder can try to impersonate Alice once or many times. We want to prevent such type of attack by using authentication. In the authentication scenario, are interesting two cases :

- when a computer is authenticating another computer
- when a person is using a public workstation that can perform sophisticated operations, but it will not store secrets for every possible user (in this case each user must remember its own secret).

Under a philosophical point of view we like to use the **closed world assumption**. When running the authentication if the process is not completely successful then you can assume it's failing. The opposite is the **open world assumption**, if you don't know it doesn't mean that the identity is false. In this case, we want to be secure and we prefer to use the closed world assumption. In this type of environment we find a third party trusted server, that distributes required information for authentication. So, in this setting it's very common that we have the trusted server, Alice and Bob that belongs to the organization, they needs to use services, the attacker is also belonging to the organization having its credentials for accessing the network enterprise. This is very common, the attacker is an insider, he has many powers respect to powers had by some external attackers. When authenticating a party is this human or a computer ? In practice there are several differences, we easily understand that a computer can store very good secret, a very long key. A computer can easily store such a secret information and run cryptographic operations. Humans are not able by definition to store in the brain long keys very complicated, they at most can afford some passwords. What is a different between a password and a key ? Since humans are very weak and very lazy, they choose as password words belonging to some dictionary or simple variations. This password can be used as a key for running some encryption, but is completely insecure. A more secure approach is to invent a password, then use a hashing function providing its hash value, and then use this value as key for running encryption. One possible attack against user authentication is the so called **trojan horse**, running the trick against the user preparing a fake login window that collects user credentials. Another type of authentication is the biometric authentication, we can examine the retina, analyze the voice, and so on. In many cases there are false positive and false negative. What is considered safe today is pairing that type of authentication with other types. When we are designing systems we need to prevent the false positive, this is the real goal (also good to prevent false negative). We start considering some real cases. For instance, a very nice setting is the one where two users, Alice and Bob, are sharing a secret key. They can use that for making a good authentication. Another possibility is that every user is sharing a key with a trusted authority or when users are using a public key. We can define many authentication techniques, like timestamp, nonce (or challenge), which is a random number to challenge the other part to do something he's expecting being able to do, or sequence numbers. We need to guarantee integrity and authentication. We start by considering the case where we implement the authentication based on a symmetric key, it means that the two parties are sharing a secret key. We want to use encryption for implementing authentication, we should be able to understand we are not using confidentiality, if we want confidentiality we must add another layer. Another important detail is that **one way authentication** is different from **two way authentication**. One way means one part is providing information about its identity, but not vice versa. So maybe Alice is giving information about her to Bob, but if she wants to be sure about Bob's identity she will need another type of authentication.

7.1 One way authentication

Lets see a schema that implements one way authentication. Alice is providing her identity to Bob (in plaintext). Bob is challenging such party by sending a nonce, means that Bob is using a cryptographical secure pseudo random number generator, sending such a number to Alice and Alice is expected to encrypt such number by using the shared key. Of course Bob can check the last message from Alice in two ways :

- one way is take the nonce, encrypt with the shared key and check with the result received from Alice.

- the other verification is just decrypt the message from Alice and check that the decryption is the nonce previously chosen.

As a result Bob is authenticating Alice, not vice versa. An attacker can collect all these messages, pairs of nonce R and encryption of R, in order to run an **offline password guessing attack**. It means the attacker knows just one pair and starts generating all keys being able to give such associations. The limits are that this case suffer from this type of attack and that it's one way authentication. There is a variant of the previous authentication scheme, in which Alice again starts with a message stating her identity. The challenge by Bob is not R but it's the encryption of R by using the shared key. Alice is expected to provide the original nonce in plaintext. In this case we need to use reversible cryptography. Of course the attacker can try again to mount up an offline attack, he's knowing a pair (plaintext, ciphertext). What is nice to observe is that the nonce R is having a limited lifetime, after that time it's no longer valid. If the nonce has a small lifetime, imagine when Bob is generating the challenge using a nonce concatenated with a timestamp. In that case he sends the message and Alice is replying but there are two interesting consequences, because Alice understand that the other part is Bob, because with timestamps Alice understands that only Bob could do that. The time interval for the validity should be very small, otherwise the adversary could try a **replay attack**. It means that, seeing a secret transmitted over the network, i don't understand such secret, i just record it and i send it again pretending to be Alice or Bob. Another case is when Alice send one simple message to Bob. In this case, Alice is just sending the encryption of the current timestamp by using the shared key. In order to do this approach, we need that the two systems should have synchronized clocks, but the clock can be subject to attacks. Then also we have to define the time limit, if we send such message maybe the validity should be one second, otherwise it could be attacked. This could be attacked but this is the base, we can improve it using extra ingredients. Suppose that Bob is the service, like a printer, and Alice should be authenticated as authorized user of the printer, this is just a string that can be sent to the printer to show that "i'am authorized to print". What if there are many printers ? They can share the same key. The identity of one printer must be concatenated with the timestamp.

7.2 Two way authentication

Now, we want to perform a two way authentication. As first attempt, lets try twice the one way authentication. First, Alice sends to Bob her identity.

Then Bob sends to Alice the challenge, she encrypt this challenge and send it back to Bob. Bob completes the authentication of Alice. Now, Alice challenging Bob choosing another challenge, and expect that Bob sends back the correct result. We can see that the actions performed are completely symmetric. This is a simple way to implement a mutual authentication. Of course we can say that there are a high amount of messages exchanged. An optimization of this approach is the following : Alice sends her identity and a challenge to Bob. Bob replies sending a new challenge with the encryption of the previous nonce. Alice answer sending to Bob the encryption of the challenge just received. We have just used three messages, but this approach suffers of what is called **reflection attack**. It's based on the construction of an extra session of authentication that is not really useful, it's just used by the attacker for obtaining good information to be used in another authentication session. Lets see the schema of this type of attack. In this reflection attack Trudy, the attacker, is pretending to be Alice. The messages are spoofed, "I'm Alice" and this is my challenge R_2. Bob responds accordingly, with the correct response that is the encryption of R_2 and a new challenge R_1. Now, Trudy is not able to respond correctly since she doesn't known the shared key, then she starts a new authentication session, saying "I'm Alice" and send the same challenge previously chosen by Bob in the first session. Bob will reply accordingly, encrypting R_1 using the correct key and will provide also another challenge R_3. Now, Trudy is knowing the correct response to the challenge R_1 and is able to complete the first session by providing this information as a response. The second session will be not completed and will go in timeout. Incomplete sessions are really frequent, so it's not sufficient for generating an alarm. We can prevent this attack by some simple tricks such as :

- we can setup a challenge rule, saying that the challenge that is generated by the initiator of the connection should belong to some set of numbers and the challenge generated by the receiver should belong to another set of numbers, for instance. This means if there is some challenge generated by one party the same challenge cannot be generated by other party.
- another possible countermeasure is to change the key, so if Alice is sending a message to Bob the key is the shared one, but is Bob sending the message.

We can see this process of basing all the challenges response authentication in a more general setting where we introduce a third party. In this case every participant in the network sharing a secret key with this server (also called **Key Distribution Server (KDC)**). Suppose that Alice and Bob shares the secret key between the server. All member of this network can be humans but also machines, programs and so on. Every user should be registered in order to use the services and the administrator should generate a new key for this user. The key will be shared between new participant and the server. Our goal is not only to finalize the authentication, but also to generate a session key to be used between the two parties. So the idea is that when we are using some authentication server, and Alice and Bob have to start a secure conversation, they need to implement the authentication to have a session key between them for some short time. Only Alice and Bob are knowing the session key, also the authentication server, and we want to be sure about that and we want that the session key is a new key every time we start a session, so it should be a random chosen key not used before. In many cases the attacker is an insider, it means he is also having a shared key between itself and key server. This is a standard insider, the attacker could be an user having a conversation with Alice and Bob. After that the attacker can store information about that conversation in order to run future attacks. The power of this attacker is to just to be a regular user of the system and may store some old session keys. The limit for the attacker are that he's not be able to guess the random number generated by the parties and he's not able to get the shared key between Alice and the server, between Bob and the server. The unique shared key is knowing is the shared key between itself and the server, because in these cases we are considering the case where the attacker is an insider. Also the attacker is expected to have a normal computational power, in fact if he sees some encrypted text, he's not able to decrypt it in a short time. Now, the question is how to setup an authentication with a trusted server ? Suppose the situation in which Alice wants to talk to Bob, so Alice sends a message to the key server saying "I'm Alice i want to talk to Bob", the server chooses the session key K and sends to Alice two messages : the first one is encrypted by Alice secret key, the second one by Bob secret key (the content of the message is the session key K). Alice can't check the content of the second message, so she decrypt the first one obtaining the session key. Now, she can send to Bob the following message "I'm Alice, i obtained this information (the second message) from the server and it's supposed to be used by you". Bob is able to decrypt this message because it's encrypted by its secret key, finds K and can send a message to Alice, like "Hello Alice, i'm Bob" encrypting the message by session key K. Alice will be sure that it's really Bob that sends this message. This scheme suffer from the following attacks :

- assume that the attacker can spoof and make MITM attack, he is between Alice and the server and between Alice and Bob. Imagine the conversation starts by sending to the service ”I’m Alice, i want to talk to Bob”, but the attacker intercept the message. He after that, can send a message to the server saying ”I’m Alice, i want to talk with Trudy”. The server chooses a session key K, encrypting one time with the secret key of Alice and the second with the secret key of Trudy, because the message received by the server is Alice wants to talk with Trudy. Now, Alice is trying to send to Bob the message like ”I’ve got the key from server, i’m Alice, this is the encryption of the session key”. Again Trudy intercepts this message, and reply to Alice saying ”I’m Bob, hello Alice” encrypting this message using the session key K. After that Alice needs a session key between she and Bob and in reality the session key is between Alice and Trudy. A first vulnerability of this protocol, is that we want to replace the identity of Bob with the encryption of its identity using Alice’s secret key. So, the server will be able to decrypt and understand who is the other party wanted by Alice. With this modified protocol, the attacker doesn’t know that Alice want to talk to Bob, because it’s encrypted by Alice’s secret key. It will become known later, when Alice will try to send a message to Bob.

- however it’s possible to run a more complicated attack in this modified protocol. Imagine that Alice wants to send a message to the server ”I want to talk to Bob”, using the encrypted identity. The attacker running the MITM attack, is able to get the message from Alice and sends to the server a different message, the message is ”I’m Alice, i want to talk to Trudy”, and this is encrypted by using the secret key of Alice. This is just a replay attack, we can just imagine that some time before Alice talked to Trudy. Trudy is storing some old message and can replays some part of the old messages. Trudy can send this information as long as in the past he talked at least one time with Alice. Since Trudy is an insider, is easy for her, can very easily push Alice to start a conversation with Trudy. In this case, the server will choose a session key and will send to Alice the two version of the session key, encrypted with secret key of Alice and Trudy. Alice decrypting the message gets the session key and sends a message to Bob ”I’m Alice and this is the encryption of the session key”. Trudy intercepts the message and understands that Bob is the other party, and she can reply to Alice telling ”Hello Alice, i’m Bob”.

7.3 Needham-Schroeder protocol

At the beginning Alice want to talk to Bob, she chooses a nonce N, sends a message to the server "I'm Alice, i want to talk to Bob, this is my nonce". The server chooses the session key K and send back to Alice the encryption of nonce, the session key, the identity of Bob and the encryption of the session key and the identity of Alice with Bob's secret key. Alice decrypt this message, check the nonce N and the identity of Bob and sends to Bob the encryption received. Bob is able to decrypt it, chooses another nonce $N^{'}$ and sends it back to Alice the encryption with the session key of "I'm Bob, this is my nonce". Alice decode this message and sends to Bob the encryption of "I'm Alice, this is your nonce minus one" using the session key K. Alice is choosing the nonce and the server replies with a message encrypted with the secret key of Alice. Alice gets the message from the server, checks information. However, the Needham-Schroeder protocol suffers is weak against the following attack :

- actually there is Trudy that intercepts the message directed to Bob, sending to him another message, which contains a different session key $K^{'}$. This session key is encrypted by the secret key of Bob, this means it's an old session key that has been actually used. Bob can decrypt and sends to Trudy "I'm Bob, hi Trudy, this is my nonce". Trudy sends to Bob the correct reply "I'm Alice, this is your nonce minus one". Now, we want to see the same attack but described in a more detailed way. We can assume that the attack is running on two different sessions, the attacker has recorded one session and the key of that session has been compromised. Lets describe what happens in these two sessions :
  - **Session 1** : Alice start the protocol by sending all the information to the server. The server replies to Alice by encrypting information by using the secret key of Alice. Now, Alice sends to Bob the ticket (the session key and its identity encrypted by her secret key). In this moment, the attacker intercepts the message, so that the message is obtained by the attacker. We can assume that later all these information are stored and the key has been compromised.
  - **Session 2** : the attacker use this key by claiming to be Alice sending the message to Bob, he replays the same message. Bob gets the message from Alice and decrypting it by finds the session key K. In this way, Bob will think that he has to talk with Alice,

> so that he replies to the message by sending a message to Alice. Again, the message is intercepted by the attacker and according to the protocol Bob is encrypting a new nonce $N^{'}$. The attacker will reply to Bob in the correct way. Now, Bob thinks that he's sharing a session key with Alice, but actually he's doing that with the attacker.

There are many possible variants of this attack. We can introduce timestamps, sequence numbers and nonce for guarantee data integrity. There is a general methodology that can be used, which uses in a smart way both timestamps and nonces, making the approach robust against replay attacks. We want to describe a more important variant of the protocol. In this variant a timestamp has been introduced and this is known as the **modified Challenge-Response variant**. In this case, Alice that want to talk to Bob, sends the message directly to Bob and not to the server. Bob chooses another nonce $N^{'}$ and contact the server sending "I'm Bob, this is my nonce" plus an encrypted message "Alice wants to talk to me, this is Alice's nonce N and this is the current timestamp t" using its secret key (also know by the server). When the server obtain this message, it understands there will be a conversation between Alice and Bob. It generates a session key K for this conversation and sends to Alice the message containing an encrypted part containing information about the identity of Bob, the original nonce of Alice, the session key and the timestamp. The timestamp has been originated by Bob. Such information are encrypted by using the secret key of Alice. The second part of the message contains the identity of Alice, the session key and again the timestamp. This part is encrypted with the secret key of Bob, then there is the nonce $N^{'}$ chosen by Bob. We don't need to sends it encrypted, because in the last message Alice is sending to Bob the encryption of $N^{'}$ using the session key K and the ticket to Bob because it's some information that only Bob can decrypt and understand. It's prepared by the server and it contains the session key so that Bob can check if the timestamp is the same and the identity of Alice. In this way with four messages we have a variant of the protocol implementing the authentication between parties.

Now, we want to see the last version of this protocol that is called the **expanded version**. Again, Alice wants to talk to Bob, Bob is generating a nonce N_B and sends it to Alice, the nonce encrypted using his secret key, so that Alice is not knowing N_B. Now, Alice sends a message to the server "I'm Alice, i want to talk to Bob, i got this from Bob, the encryption of a nonce, this is another nonce N_1 generated by me". The server will now generate a session key K_{AB} between Alice and Bob and decrypting the message encrypted by the secret key of Bob finds the nonce N_B. Subsequently, the server sends a message to Alice containing several parts of information (the nonce N_1, the identity of Bob, the session key, and the ticket (the identity of Alice and the nonce N_B, this message is encrypted with Bob's secret key)) and the whole message is encrypted using Alice's secret key. Alice decrypts it and got the nonce she sent to the server, the identity of Bob, the session key and the ticket to Bob, so that Bob can check it and realize if it contains the nonce previously generated. After that, Alice sends the ticket to Bob and sends the encryption of a new nonce N_2 using the session key K_{AB}. Bob can decrypt this information using the information contained in the ticket, and can reply to the message encrypting a variant of the nonce $N_2 - 1$ and another new nonce N_3 using the session key. Alice can sends some confirmation sending the encryption of the new nonce received, encrypting it with the session key. This last message isn't necessary, because after the 6th message the authentication is already implemented. This is a protocol inspiring the way the Kerberos protocol is implemented.

7.4 Kerberos protocol

Kerberos is a general framework for providing authentication in distributed systems and is used in network enterprise, but not only for enterprise because in any case people are using Kerberos for LAN, so what is important to say is that this solution is implemented in a good way for all operating systems. Kerberos is considering the problem that Alice wants to access a service provided by Bob, to do that we must authenticate Alice, she can use the service, in some cases also Bob should be authenticated. In some cases the conversation can take several messages, in those cases is important to have confidentiality, to have the possibility of having the session key between the two parties. Kerberos is introducing the trusted server and this server is a trusted authority and the authority is of course a reference server in the Kerberos framework, trusted by every part (KDC). This name comes from the fact that it's the authority distributing the session keys when they are requested. There is the KDC that stores a master key for every user in the Kerberos network. Every participant of the Kerberos network have a master key, configured when you are registering to the framework, and this is a secret key that only the KDC and the participant know. The approach is to generate tickets allowing users to access services. A ticket contains some information that are provided from one side to the other side and the entity that provides it is not able to understand its content, only the receiver can decrypt it and check the session key and the identity of the person. This approach uses also ticket valid for a given time window, so you can decide different expiration time for different types of service. The main idea is that Alice obtains from the KDC a ticket and then she will use the ticket for the conversation with Bob, this is one of the versions of the protocol. All messages exchanged are safe with respect to confidentiality and data integrity, and they are basic requirements of information security. A framework like Kerberos provides security for applications well designed in some unsafe mode, like telnet. Then the KDC will provides some reply, that Alice will use to talk with Bob. This information are provided by the trusted server, that is encrypted by the master key of Alice and such information are the session key, the ticket to Bob, which contains the identity of Alice, the session key and other information that can be used in several cases. Alice sends something to Bob in order to implement the setup of session key between she and Bob. The ticket can contains other information like not only a session key, not only the identity of Alice but also the timestamp, lifetime of the ticket, an important information that defines the expiring time, starting from the timestamp the ticket will be valid for lifetime period. We understand that in order to have a good implementation we need all principals (i.e. users) have a synchronized clock. This can be a vulnerability. Attackers can try to attack the clocks for making particular smart attacks violating the rules by lifetime. Alice when gets the message can checks the nonce and learns about the lifetime. When asking for the service, Alice sends to Bob the ticket and a new information that is known as the **authenticator**. It is the identity of Alice and the timestamp generated by Alice, encrypted using the session key. Bob when decrypt this ticket, gets the session key and other information like the identity of other party and he can decrypt the authenticator and checks the identity of Alice, also checks if the timestamp is consistent with the current time known by him. Now, Bob sends to Alice the encryption of the timestamp sent by Alice using the session key. Alice can understand that the other party was able to decrypt the ticket, thus implements the authentication on Bob. Now, we want to discuss some practical problems. Lets imagine the situation in which Alice asks to the server several tickets, how can we manage this case ? We have two possible solutions :

- every time that Alice needs a server she provides the password, the workstation of Alice derives the master key from the password and encrypts the message. This means that for any possible service she has to retype the password.

- another solution is that the master key is stored locally. This is considered very insecure.

None of the previous solution are considered secure. This introduces a new type of solution based on **meta ticket**, but in the Kerberos world people are using **Ticket Granting Ticket (TGT)** mechanism. The main idea is, Alice has a ticket to ask for tickets. Every time she needs a ticket she's not requested to retype the password, she just show the ticket. To implement a TGT we can imagine some particular moment, in the morning Alice gets to the office, she opens the workstation and performs a login, she needs to type the password and from this password some messages are exchanging and a session key is derived for Alice. This is a new session key, it's used for a session. For several hours the work session of Alice is continuing, using this session key. The session key have a fixed lifetime. While doing the job we give Alice a TGT, including such session key, and other useful information. If the server wants to give Alice a special session key, the server will encrypt such session key using some master key. Later, Alice sends request to the server saying "I want to talk to Bob". When she sends such request, she will send also the TGT, the ticket for having the ticket, since this is a ticket. Alice gets TGT containing several information, in particular such TGT is encrypted by using the KDC master key, so that when Alice want to use server V the KDC will obtain the TGT and will generate a ticket. We can image the Kerberos framework like an office providing services and there are two special offices, because there will be an office making the authentication of all users (**Authentication Server**) and then another office providing tickets (**Ticket Granting Server**). This scenario is constituted by the following steps :

1. Alice logging on the workstation and request the service. It will send a message to the Authentication Server (TGT request), and the reply for this message will contains the ticket plus the session key. We notice that these two messages are sent once per user login session.

2. The server performs the authentication in some way, like accessing some local storage for checking the identity of Alice, and so on. If the result is positive, it will provide to Alice the ticket and the session key.

3. Now Alice want to use a particular service. In order to use this service she sends a message to the office for obtaining the ticket for the service. To do that she has to show the TGT and the identity of the service

she want to use. So, she sends a message and obtains a reply, where the reply is a ticket for the service and a session key to be used in the service. This type of message is once for type of service.

4. TGS decrypts ticket and authenticator, verifies request, then creates ticket for requested service.

The first message is ”I'm Alice and i want TGT”. It's the AS_REQ. The AS generates a session key S_A for Alice, then it finds the Alice master key and provides to her (AS_REP) the encryption of of the session key S_A and TGT, which contains the identity of Alice and the session key encrypted using the KDC master key, using Alice master key. Now, Alice want to use a service from Bob. She needs a ticket for that, so that Alice asks for a ticket to Bob. This is a request to the office providing tickets (TGS_REQ). The message is a message like ”I'm Alice i want to talk with Bob, this is my TGT”. Alice is using a ticket now, it's the TGT but it's a ticket, but she needs to provide also the authenticator, some information (like the timestamp) encrypted using the session key. At server side, the server generates the session key K_{AB}, do all the checks, then finds the master key of Bob and prepares the ticket for Bob, containing the session key K_{AB} and the identity of Alice, encrypted using Bob master key. The reply of the server (TGS_REP) is encrypted by the session key generated for Alice and contains the identity of Bob and the ticket for Bob. For using a ticket Alice is sending a message to the service, AP_REQ, which contains the ticket to Bob and the authenticator (timestamp encrypted with the session key K_{AB}). Bob decrypts the ticket, gets the identity of Alice, the session key and extracts the timestamp and can reply to the parties, with an AP_REP (typically the value of timestamp plus 1, encrypted with the session key), so that the authentication is performed. The authenticator is used to avoid : replay of old messages sent to the same server by the adversary,replay to a server. It doesn't guarantee data integrity (MAC is required). The KDC and TGS are the same ? It's a standard considering the protocol and in this case we have many KDC, there is a master KDC and all updates are made on the master KDC, that is pushing new information to the other KDC. KDC stores information most frequently used in read mode (TGT and tickets).

7.4.1 Realms

There are several cases where you can imagine different networks belonging to different enterprises that for some reasons need to cooperate. It means that some users of one network may want to use a service from another network. If you want to implement some integration between the networks you can do that and one way of doing that is implementing **realms**. Realms is like a federation of Kerberos networks, with some strong properties, you can't just take two Kerberos network and say "ok lets make it realms". It's not so easy, you have to redesign, reconfigure some parameters about the networks. We can setup different realms, but you can't setup some whatever number of realms, because the cost of setting up k realms is quadratic in k. Each realm has a different master KDC. What is important to understand is that if you have two realms, your master KDC should be a principle of the other realm and viceversa. This means there should be some key agreed between the different master KDC. When we want to add a new realm, happens that the master KDC of the new realm will be a new principle of yours, we have to setup a key shared between the two parties. When we are setting up two different realms we want that whatever user in the two realms can use services from the other realm, we want a completely symmetric situation. What is important to say is that to use a service in Kerberos we need to be authenticated. The authentication is done from the KDC of the same realm, since every KDC store information about the local user in that realm. We need to setup some trust between the different KDC servers. A KDC server will get a request coming from another realm after that a user has got the authentication with its KDC server. So, the authentication is made locally, after that the user is enabled to ask for a service in some other realm, the authentication is finalized in the local realm of the user. This is a simple schema, we should understand now that if, in some incremental way, we will let our network grow in time, add a realm, for every realm added we need to register the new KDC of the new realm in all our KDC, we will have a linear number of registrations and keys to be managed for every new realm, this is why we will have in total a quadratic effort.

7.4.2 v4 vs v5

The previous scenario is not so nice, we can consider it like a limit of Kerberos and this is the reason why while passing from Kerberos 4 to 5 also a hierarchy of realms has been introduced, so that isn't necessary to have a whole knowledge of all possible realms. We also should remember that at time of designing of Kerberos 4, the encryptor employed was DES algorithm (nowadays completely broken), so what happens while passing from Kerberos 4 to 5 ? In Kerberos 5 has been introduced also nonce during the finalization of authentication, not only timestamps allowing to make a finer contrast to attacks, because there is a simple way we can exploit it.

We can organize our management of nonces and timestamps so we become completely robust against replay attacks. Also we can encrypt information by using different algorithms. There is some flexibility, so that we can add extra features without the need of redesign the protocol. One of the most nice thing introduced is the concept of **delegation of rights**. Alice can allows Bob to access her resources and Alice can define in a very precise way not only the amount of time Bob is allowed to use them, but also what subset of resources are granted to Bob. It has been introduced also the concept of **renewable ticket**, so it's easier to prolong the duration of a ticket.

7.5 Public key based

If we want to setup authentication based on public key, we can send a signed message that contains some challenge, password, whatever we want. The weakest point of this approach is can we trust the public key ? Because when somebody receives a digital signature it must be possible to verify that the digital signature is trustable. We just get the message, we get the digital signature, the verification is based on two steps, we just compute the hash and the second step is to decrypt the digital signature by using the public key and check if the result is the same we got from the hashing. So, we are aware about the fact that in order to run the verification algorithm, Bob needs to know the public key of Alice. What if the attacker sends a message signed by yourself and says to Bob, this is a message signed by Alice, this is Alice public key, instead it's the public key of the attacker. If Bob believe to this statement he takes the public key, makes the check and finds that public key is associated to the private key for signing the message. We need to associate in a secure way a public key with some identity, we need a strong association. One tool for implementing such association is a **digital certificate**, that can prove a digital identity or also some analogic identity. Which in turn it belongs to the **Public Key Infrastructure** approach.

7.5.1 Needham-Schoreder

This scheme is somewhat simulating the presence of some authority that is providing public key of users. This authority is trusted from all parties, so that the public key we got from such authority is the correct one and everybody can ask for the public key of everybody. In order to implement a mutual authentication between Alice and Bob, Alice could ask the authority "I'm Alice, I want to talk to Bob". The authority replies, sending to Alice the identity of Bob, the public key of Bob and the digital signature of the previous fields. So, Alice get this message and check the digital signature using its verification algorithm. She generates a nonce N and sends to Bob the encryption of the nonce and its identity by using the public key of Bob. So, Bob can decrypt the message by using its private key and knows the nonce and the identity of Alice. Now, Bob want to check Alice, so Bob send to the authority "I'm Bob, i want the public key of Alice" and the authority provides the public key and the identity of Alice, and the digital signature of these fields. Then Bob can check the digital signature, so he gets the public key of Alice, Bob can generate another nonce $N^{'}$ and send to Alice an encrypted message containing both the nonce generated by Alice to prove it's me and the new nonce just generated. Alice can decrypt the message ,check if the N is the correct nonce and can send back a message to Bob that is the encryption of the nonce generated by Bob by using Bob public key. We have not assumed that the user can be cheated about the public key. We are assuming that the trusted server is providing the correct public key. How can we attack this scheme ? We assume the attacker is another user of the same network, so just like Alice and Bob, and he can talk to the trusted server using the same way to make secure authentication of all users. The attack is based on the construction of two different sessions R_1 and R_2. R_1 is the authentication between Alice and Trudy, and R_2 is the authentication of Trudy pretending to be Alice with Bob. This is a way to implement MITM attack. As a precondition is needed that Trudy is able to push Alice to start an authentication with herself. It's not so hard, because if they are users in the same network they are **coworkers**, and so there are many processes that are somewhat relating to the work activity of both parties. In R_1 Alice starts a conversation with the attacker, obtaining its public key. Alice sends a message to the attacker containing its identity and the nonce N encrypted by using the public key of the attacker. In R_2 the attacker pretends to be Alice and wants to talk with Bob, so wants to cheat Bob about the identity. So, the attacker sends to Bob a message that is a nonce and the identity of Alice encrypted by the public key of Bob. Bob receives this message from the attacker and replies, he's thinking to send a message to Alice, but he's actually sending a message to the attacker. Now, what is required by the protocol is sending back a message to Alice. So, Bob generates a new nonce $N^{'}$ and sends back to the attacker the encryption of the nonce received and the new nonce by using the public key of the Alice. Now, the attacker can continue in the first session of authentication between Trudy and Alice and since the last message Trudy obtained by Alice was the encryption of the nonce and identity of Alice, now Trudy can reply and sends to Alice the nonce obtained by Alice and the nonce generated by Bob.

According to the protocol, Alice is replying to the attacker sending as a response the encryption of $N^{'}$ by using the public key of Trudy. Trudy gets this information and now can act like Alice by implementing the last step of the other session, sending to Bob the encryption of $N^{'}$ by using the public key of Bob. How to fix this attack ? The protocol is just the same, but when we finalize the authentication if we want to send a message from Bob to Alice, sending back the nonce generated by Alice, the new nonce generated by Bob, we need to sends also the identity of the sender. So the attack no longer works, this is just the only difference that we need for contrasting this type of attack. The previous attack now is failing, because when the attacker ask to Alice to decrypt a message to obtain $N^{'}$, Alice gets the message but this time the message contains the identity of Bob, Alice is talking to Trudy, so Alice is able to say this is not your identity. This is the identity of Bob. The attack fails for this reason.

7.6 X.509 authentication standard

A digital certificate is a message that assure that the public key of an entity really belongs to that entity, since it is signed by a trusted authority. Some standards for writing digital certificates has been developed, the current standards is X.500 at version 3. The design of this standard is belonging to a wider process of designing a larger standard, the standard X.500. In such standard three examples of authentication protocols have been proposed :

- **One-way authentication** : this is the simplest case, the standard X.500 defined a one way authentication, a simple message, being sent by Alice to Bob, after this message Bob will authenticate Alice. He knows the identity of Alice. This message contains the digital certificate of Alice, this is like sending a secure public key, the timestamp, the identity of Bob and the encryption of the session key by using the public key of Bob, and the digital signature of the previous fields. Bob can use the digital certificate for extracting the public key and check if the digital signature is ok. In positive case, Bob can trust the message. This is not symmetric authentication, the role of the client and server are different. Bob is the server, we can expect the public key of server is known for several possible reasons. Since the two sides are not symmetric we expect is easy to know the public key of Bob, but not easy to know the public key of Alice. The attacker cannot just record the information and use them with another server, that information is containing the identity of Bob and encrypted information using the public key of Bob. The timestamp is somewhat preventing the reuse of the message multiple times. The attacker can record the message and send it to Bob pretending to be Alice that wants to start a new session. The presence of the timestamp is somewhat preventing that, but we want to describe a technique that is a general technique showing how to use a combination of timestamps and nonces. We can use both of them to contrast replay attacks, based on the fact that the attacker is storing your message and using them again after some time. Assume that we want to use only a timestamp, so Alice is sending in a secure way some information to Bob, and in this information there is also a timestamp. We don't expect Bob is receiving in real time this timestamp, even with completely synchronized clocks. So, we need to define a lifetime for the validity of this message. If Bob receives the message in this time interval then the message is valid, otherwise he will say it's a replay attack. If we are in a very fast network, the attacker may replay very quickly this message in the correct time window. So, in order to prevent that we can add a nonce. The nonce has a particular role, because when the server obtains such information the server can say ok i get the message, the timestamp is good, i'm still within the interval, i store the nonce. The nonces are not to be stored, because there are too many nonces. It's not a good idea to store them. We store a nonce, if the server is storing a nonce the replay attack will be blocked even if the attack is very quick. He will send a message some time after the message of Alice, containing the same nonce. So the timestamp is valid but the nonce not anymore, and the server can see the attack. We ask server storing nonces. The nonces will be stored by the server just for this interval to recognize the attack. This is a very good measure to contrast replay attacks.

- **Two-way authentication** : there will be a message from Alice to Bob and from Bob to Alice. The messages are naturally enough, there is a signature of the message, a timestamp, a nonce N, the identity of Bob and the session key (encrypt with Bob public key) and a digital certificate, so it's possible to check the digital signature. Notice that the other party is also sending a similar message, digital certificate of Bob, message coming from Bob and digital signature made from Bob on the message. The message is changed at the following one, when Alice is sending a message to Bob there is a timestamp, nonce, identity of Bob and the proposal of session key k encrypted with the public key of Bob. The answer provided by Bob is somewhat similar because there is another timestamp for time-stamping the moment, a new nonce $N^{'}$, the identity of Alice, the nonce originated by Alice and the proposal of another session key $k^{'}$. encrypted by the public key of Alice. With two session keys it means there is a precise role. From one side to the other side with a session key, messages in the opposite direction with another session key, this can prevent attacks based on reflection, for finalizing the first section.

- **Three-way authentication** : there is also a third proposal that is based on three messages implementing mutual authentication, meant to be robust against replay attack. The three messages are like this, Alice to Bob, Bob to Alice and again Alice to Bob. In the first message Alice is providing a message, digital certificate and signature of the message. The message contains several components, has the proposal of a session key k, identity of Bob, a nonce and a timestamp. When the timestamp is described with symbol O it means it's optional, the protocol is admitting the possibility that is not provided. The reply from Bob is similar, a message, the digital certificate, signature of the message providing an optional timestamp, the nonce obtained by Alice, a second nonce $N^{'}$ chosen by Bob, the identity of Alice and a proposal of a public key that can be the same, so this is not really a proposal but it's a confirmation, it can also be a different session key if they want to implement symmetric encryption based on different symmetric keys. The new message being added to the protocol is a message from Alice to Bob, which contains the digital signature of : two nonces N and $N^{'}$ and Bob identity, and the Bob identity.

They are somewhat obsolete, they have been proposed several years ago and developers and security analysts are proposing different variants for implementing the authentication. Now lets consider the following exercise. Suppose that Bob sends a nonce to Alice, and she replies by sending the digital certificate, another nonce, the nonce chosen by Bob, the identity of Bob and she digitally sign the two nonces and the identity of Bob. Now, Bob can send a message to Alice that is somewhat similar, digital certificate of Bob, nonce by Bob, confirmation of the nonce obtained by Alice, identity of Alice, and the digital signature of the two nonces and the identity of Alice. Some Canadian researchers found a way to run an attack named as **Canadian attack**. In this scenario, the attacker pretends to be Bob and send to Alice its nonce. Alice replies to the attacker, she think is getting the message from Bob, sending certificate, nonce, identity, signature. Now, the attacker pretends to be Alice and sends to Bob a message containing the nonce previously received by Alice. Bob think he's talking with Alice and sends the digital certificate, nonce N_B, N_A, the identity of Alice and the signature. Now, the attacker sends a message to Alice pretending to be Bob, a third message of the previous session of authentication, adding certificate of Bob, all the nonces, the identity of Alice and the signature made by Bob to confirm some information to Alice. This is another successful attack.

7.7 Public Key Infrastructure

Official PKI is based on the presence of a trusted **Certification Authority (CA)**. What happens if a legal CA is used for certifying the public key of some user and such a certificate is sent abroad in another country where that CA is not considered to be legal ? This is a big problem, because if the issue is unknown, there is no path of trust between such unknown issue and some known and trustable issuer, you have to decide about your trust, that issuer of a certificate. This is up to you to trust or not trust, there isn't a general rule and all suggestions depends on the level of security that you want. We have two types of certification authority, the official one, registered in some office, and the unofficial one, the one you are just creating in some protected environment, because all the participants in the environment are knowing the public key of this CA and may be you can hard code the public key of the local CA in the main host of your network. In general, digital certificates issued by an unknown issuer are considered useless, if you don't know if you can trust. Other important details are associated to the keys that are used for public and private keys because you know that after time is passing you need longer keys, so a digital certificate should expire.

7.7.1 X.509 certificates

The standard X.509 specifies how a digital certificate should be prepared, its structure. A digital certificate contains some standards fields and the role of any field is well defined. The version is the first field of the digital certificate and the current version is version three. Certificate serial number is an integer number (the concatenation of the name of certification authority and serial number is unique). The field signature saying what is the algorithm used for computing the signature on this certificate. Issuer, the name of the certification authority that creates the certificate (not just the name, a standard name described by the X.500 protocol that is also stating how to write names of digital entities). In X.500 we find a country, an organization, the organization unit that is a particular organization that is associated to a digital certificate, the common name, it's like a nickname used for describing the subject in a short name. There are many rules about what types of names are allowed, what type of names can be sub-name, OU is allowed under the name present in O, and so on. What is nice to say is that it has never invented an official way to describe a X.500 name, so different application display them differently. Validity contains two subfields, one subfield is not valid before this date and the other is not valid after. In other words, validity defines an interval using two dates. Then there is the X.500 name of the subject. Then subject public key info field, which is constituted by two subfields, an algorithm identifier and the public key of the subject. Then we have Issuer unique identifier, a code associated with issuer, subject unique identifier, extensions, all such information are signed by the certification authority.

7.7.2 Hierarchy of CAs

A digital certificate can be revoked for several reasons. When we are checking our digital certificate not only we have to check the certificate but also we should check if the certificate is revoked or not. Now, lets talk about how Alice and Bob can check the validity of a digital certificate. What if two parties are using different CAs ? In the digital certificate released to Alice the issuer is the CA to which Alice is referring to, subject is the owner of the public key (Alice), a field saying that Alice is not a CA and the signature of CA of Alice. Now, CA trusted by Bob can release a digital certificate for CA trusted by Alice, CA of Alice is a client of the other CA. What we have is a **hierarchy of certification authority**. So, in this way we can establish a certification path in order to accept digital certificates issued by unknown authorities. This path can be infinite ? The answer is no, because at some point we should stop, there aren't infinite CAs. The certification of the last CA can be one of the two following types :

- **Self signed certificates** : the authority is not only signing, also signing a statement saying what is its public key.
- **Mutual trust** : the case where CA A trusts CA B so it's issuing a digital certificate and vice versa. In other words, they are providing a double certification.

These approaches are not so efficient, since in the worst case we need to traverse all the paths until reaching the CA that we trust, then we can stop. There are three cases of **certificate revocation** : the private key is compromised, user is not certified by CA or when the CA is compromised.

There is what is called the **certificate revocation list**, which is a list that contains a list of serial numbers of certificates that have been revoked. This is very important, CAs are publishing a revocation list and if we want to check if some certificate is still valid we need to examine some original source of information, so one possibility that was meant when designing system is to use such revocation list. There are several entries and for each entry there is a serial number of some certificate, the revocation data and some extensions. It's important to say that the last field is the signature, made by the certification authority on all the content of the digital certificate. There are two interesting fields, there is a data informing when the current list has been issued, and another possible field that is optional, which contains information about the new date for the next publishing of another up to date revocation list. The revocation list is a big document, not just like a standard X.509 certificate because contains many entries. This is considered to be like a drawback of this way of informing the users about revocation of certificates. Managing such list is not so effective, is somewhat demanding in terms of resources, delay while making connections. In version 3 certificates have much more information like the url where a browser can connect for using some services provided by the CA for making a query. The browser is not looking for the whole revocation list, just making a query to the CA, done accordingly to some online certificates status protocol. The certification authority can collect data about users, and is able to associate information coming from the connections. So, the CA is able to check and profile all users connecting to the websites using certificates issued by the CA, this means a concern from the beginning and researchers have addressed this concern and proposed some changing protocol known as **Online Certificate Status Protocol (OCSP)**. Keep in mind that in the web, digital certificates are the main tool for making authentication of one party, because when you connect with https protocol your browser will authenticate the server, because the server is offering information about itself based on a digital certificate. Your browser check such information, the server is implementing a sequence of steps (TLS handshake), and will be able to authenticate the server. **Pretty Good Privacy (PGP)** allowing us to implement secure email, used at application level. The idea of the creator was "i don't like CAs, they can be attacked, so i prefer a distributed approach". He proposed the so called **web of trust** concept. It's something that is proving the correctness of the association between a public key and an identity, not by a digital certificate but by using the trust obtained from other users. In this way, a public key is signed by a lot of users, and we can assume that the public key is good.

7.8 Password based

Instead of storing the password, we can store some information derived from the password, may be its hash value, so we receive the password by Alice using a secure connection, we compute the hash and check if we are storing that hash. This approach isn't so good, because attackers can run what is called **dictionary attack**. They take a long list of words, coming from official list of words that we can find in the web for implementing dictionaries. We import text files and can compute the hash of all possible words. The important point is that a password must not be sent in clear, or also in an encrypted way, because an attacker can just store the encrypted password, and authenticate itself sending this encrypted information. This means that what is sent for making authentication should be different every time, otherwise replay attack will be easy for an eavesdropper. Lets talk about the way Unix is storing passwords, at least in the original approach. Originally, Unix systems were considering just the first 8 characters of the password, what is coming after the 8th character is useless. The password chosen by the user is converted to some secret key K, at that time the encryption algorithm was DES. This is why they choose to use the first 8 characters (ASCII encoding). ASCII is a 7 bit standards, so we got $7 \cdot 8 = 56$, that is the same number of bits for DES key. Once determined a DES key, we compute DES of a sequence of zeroes, and again DES for 25 times. The results is stored at server side. The encryption of the password is stored in a file that is not public readable. Every user able to read this file can run a dictionary attack, we get the DES key and try to encrypt the sequence of zeroes, then we can check for the password. If we find a match, we find the username and the job is done. In order to prevent the efficiency of such type of attack **password salting** has been introduced. When the user is storing the password because he has just chosen a password, the system generates a random string, and concatenate it with the password. Before analyzing the consequences, we should say that the salt is saved as clear text, so is known also for the attacker. The attacker can no longer choose a word from the dictionary and check if there exists some user that want to use that password, due to the salt. So, the attacker should, for every user, takes a word from the dictionary, adds the salt and checks the encryption. This is reducing the efficiency of the attack, but it's not a complete fix of the problem, is just offering some constant factor to the time running of the attack. Suppose that Alice want to authenticate herself to Bob, she can't send the password in clear, cannot try a DH exchange for establishing a session key due to MITM attack. Maybe she can try a challenge response handshake, but the eavesdropper can carry out a dictionary attack again. Alice can choose a word that doesn't comes from a dictionary, but it's very difficult and few people can do that. A possible alternative is to use a strong password protocol knows as **Lamport Hash**. The goals are the following : obtains the benefits of cryptographic authentication with the user being able to remember passwords only, no security information is kept at the user's machine, someone impersonating either party will not be able to obtain information for offline password guessing.

7.8.1 Lamport hash

Lamport invented the following protocol. Alice is choosing a password, Bob is the server that needs to store the password. Bob instead of storing the password stores the name of Alice, a number n, and the hash of the password computed n times, with some hashing function h. When doing authentication Alice sends the username and says "I'm Alice", Bob replies sending the number n. Alice write the password and on the Alices workstation is computed the hash of the password $n-1$ times. This string is sent to Bob. Bob gets the string, computes again one level of hashing and check if the result is the same as what is stored in the database. If yes, the authentication is successful and the line in the database associated is updated, decrementing by one n and the hash n times is replaced with the received one (hash $n-1$). Every time Alice is sending a different hash, so this contrast the replay attack. Of course this is based on the security of the hashing function, it's difficult to go to the counter image of some hash, we need a cryptographic hashing function. Also no dictionary attack is possible, because it's difficult to go back to the previous version. However, it's still possible some attack to the server. The strength of this approach is sending one password one time only. We can use salting for this approach, we can store the password hashed many times concatenated with a salt, when n reaches one just change the salt (i.e. generate a new salt). Also it's possible that Alice is using the same password with different instances of the server, without using different salts, and again salting is somewhat making harder the dictionary attack because in case of no salt the adversary can take a word from the dictionary, compute all the hash and checks if there is some user matching. In the case of salt, the adversary should do that for every single user because he has to concatenate the word chosen with the salt. However, there is one attack to the Lamport hashing called **small n attack**. Imagine the case where the attacker is making MITM, so that Alice is trying to perform the authentication, the attacker is forwarding the message to Bob, Bob replies with some number n, the attacker forward the reply of Bob to Alice by changing the number n and making it smaller. Alice will reply hashing a limited number of times. Now, the attacker knows what is needed to make authentication, just computes extra level of hashing and providing the result to the server. The n attack is very bad, because if we want to prevent such attack we have to adopt some strong measures. One possibility is that Alice is knowing the value of n, but this is a very hard requirement for Alice, she should remember this extra information that changes every time. Another possibility is to use one client only, because in this case the client used by Alice will be able to store the value of n and checks if the next n is the good one. This approach allowed the construction of the so called **S key**, token generators and they provides this approach for secure authentication standardized as **One-time password** system.

7.8.2 Encrypted Key Exchange

In this approach we have a requirement, user and server are sharing a secret. This secret can can be also a weak secret, because next actions will be providing more security, in order to be robust against the dictionary attack. It provides a mutual authentication and a session key. Alice is having some password, and there is some rule (like a hashing function h) allowing to derive some information from the password ($W = f(pwd)$). This is the shared secret between the two parties. Bob is not knowing the password, just the value of the function. Alice doesn't need to remember this value since she just have to remember the password. The first message is sent by Alice that starts the conversation, sending to Bob its identity and the encryption of $g^a \bmod p$ (like in DH) by using the value of the previous function as key (i.e. W). Bob choose a random number b and a challenge c_1 (a nonce), and reply to Alice encrypting $g^b \bmod p, c_1$ using as key W. Both parties know they can build the session key K computing $g^{ab} \bmod p$. In order to construct a session key, the two parties must share a secret, otherwise is not possible to construct it. Now, the two parties need to confirm the authentication, Alice sends to Bob the challenge c_1 and another challenge c_2 originated by her, encrypted with the session key K. Alice expects to receive back a reply from Bob that is the encryption of c_2, made by the session key. In this way we are making more robust DH approach against MITM attacks, because the attacker doesn't know the weak secret W. Also the dictionary attack is ineffective, because even if the chosen password is weak, the choice of the random number a doesn't allow the attacker to compute g^a. If the attacker knows the password of Alice, he can impersonate her, but he doesn't know the random numbers chosen by Alice and Bob (he's not able to know the session key). The second part represent the authentication, but also in the first phase we are performing an authentication using the shared weak secret W. In the second case, we are performing an authentication based on a strong key K. If the password file (information stored at Bob side) is stolen by the attacker, theoretically speaking is possible to run a dictionary attack. For this reason, researchers are talking about the **augmented EKE**. The idea is that Bob is storing some information that is derived from the password of Alice, such information is used for verifying the password but at Alice's side it's necessary to know the password. If the attacker knows W, he can impersonate Alice, but if we add augmentation we add some extra property, we need to introduce some extra ingredients so even if the attacker is knowing W, this information is not sufficient to impersonate Alice, because to do that he needs to know the password of Alice. Alice and Bob still shares a secret W derived from the password. Bob also stores Alice's identity, Alice's public key and the encryption of Alice's private key using a new secret W' still derived from the password, that we will call it Y. At this point Alice chooses a random number a and send to Bob its identity and the encryption of $g^a \bmod p$ using the shared secret W. Bob chooses another random number b and sends to Alice the encryption of $g^b \bmod p$ using the secret W, the encryption using the session key $K = g^{ab} \bmod p$ of Y, and a challenge c. Now, Alice decrypt the first message and compute the session key K. After that she is able to decrypt the second message to figure out Y, compute W' from the password and decrypt the previous result in order to get its private key. Finally, Alice sends to Bob the hash of the session key K and the challenge c using its own private key and Bob can check it using Alice's public key.

7.8.3 Simple Password Exponential Key Exchange

It's a variant of EKE. Its main idea is using W in place of g. We expect that g is knows, because this is a DH exchange, we define what is the private and public key. In this way, we can use W in pace of g, so we are going to transmit from Alice to Bob $W^a \bmod p$, and $W^b \bmod p$ in the opposite direction. Now, Bob and Alice can compute the session key $K = W^{ab} \bmod p$.

7.8.4 Password Derived Moduli

Another EKE variant is PDM, where what is interesting to notice that p is no longer a fixed number, but it's a number that depends on the password.

In this case, we fix $g = 2$, because p is a function of the password. This is the password derived.

Augmented PDM What is storing the server for Alice ? The name of Alice, number p and $2^W \bmod p$. W is again some hash of the password of course, in this case Bob is not storing it, but some information derived from the hash of the password. Also is storing p, but what we said the number p is derived from the password (the server doesn't known the mechanism for repeating the process of generating p). With these preconditions Alice can choose a random number a and from the password can compute the weak secret W and number p, using some algorithm that is again based on some hashing function, in this case the hash value is modified so that the final number will have some extra requirements (the best case is when p is prime). Now, Alice can send $2^a \bmod p$ to Bob. Bob can reply with $2^b \bmod p$. In order to make the approach stronger, Bob is also sending the hash of the session key $2^{ab} \bmod p$ and $2^{bW} \bmod p$. This latter value is computed by Bob, but Alice is not able to compute such a value, she's not knowing the number b chosen by Bob. So, Alice replies with another hash of the same arguments previously described. A possible variant is instead of using DH with the operation well known, for example RSA verify operation. Bob stores the name of Alice, the hash of the password W, the public key of Alice, and another information Y, which is the encryption of Alice private key using $W^{'}$. This value is another value derived from the password. Alice starts by choosing a, compute W from the password and sends a message to the server, which is constituted by Alice identity and the encryption of $g^a \bmod p$ using W. Bob is able to decrypt this information. Now, Bob choose a random number b, a challenge c and sends a message to Alice, which is the encryption of $g^b \bmod p$ using W and the encryption of Y using the session key $K = g^{ab} \bmod p$, and the challenge c. Now, Alice is able to decrypt $g^b \bmod p$, can build the session key and by using it, she can find what is the original value Y. Alice can generate the other weak secret $W^{'}$ for decrypting its own private key. In fact, Alice doesn't storing locally the private key, she gets it from the server. Then Alice send the following message to Bob : the hash of the session key and the challenge signed with Alice private key. An adversary cannot impersonate Alice even if he is knowing the W. If the adversary is knowing the original password, then he can impersonate Alice. This is a fail of the approach, if your authentication approach is password based, then you should understand that if the adversary manages to find the password he will be able to impersonate the user. For this reason, today we are considering **multi factor authentication**, which is an authentication method based on two different steps.

8 IPSec

We want to talk about IPSec using what the engineers call **black box analysis**. It means, that we need just the knowledge of the services provided by the protocol and how to interface our application with the protocol. A black box analysis is just some consideration bout the protocol by observing what are the requests we make to the protocol and what are the answers to our requests, we don't know how the box is organized inside. Lets start considering the **classical protocol stack**, which is constituted by the application, transport, network levels and then we have lower levels. Naturally, we can introduce security between application and transport level or also between transport and network and transport level. The consequences of the previous two approaches are that the datagrams are encrypted. An attacker could use a **traffic analysis attack**. Indeed, also if the packets are encrypted, may be the attacker could understand you are using a browser and accordingly classify the type of traffic originated from some target. If he's able to determine the port number, then he can guess what type of application we are using, allowing traffic analysis. When the attacker starts what is called **deny of service**, he will try to attack our network, he needs to know the type of packets going to the network. If he has run some traffic analysis, he will know the type of packets and can address some sub targets. If 90% of traffic is of type A i prefer to attack type A, if i want to run my deny of service. The IPSec approach provides security at datagram level. When we are ensuring security for datagrams we are talking about one datagram. We know that a datagram is composed by a header and a payload. If we want to encrypt information we can't encrypt the header, because it contains the destination address, or other information like TTL. So, we may decide to encrypt the payload. If we implement encryption for datagrams we have extra effort for routers, they are receiving datagrams and retransmitting datagrams to next routers, if we use encryption or other operations we ask routers to work more. IPSec mechanism provides three important security services :

- **Authentication** : we are able to understand who the two parties are. We are authenticating the two parties showing the source and destination IP address. It implies also data integrity, being able to understand what are the two parties.

- **Confidentiality** : the traffic that can be potentially capture is encrypted.
- **Key management** : it's a standard for generating session keys for communicating.

Naturally, we can choose what security goal we need to address, for example we are interested only on providing confidentiality and not key management. There are several documents describing the IPSec standards, very complicated. In IPv6 IPSec was originally mandatory, then the IETF decided to make it optional. Some benefits of IPSec are the following : we can setup our firewall/router in order to provide strong security for the traffic that is going from outside to inside or vice versa, we cannot bypass the gateway running the IPSec, because we have no possibility by using some other measures. IPSec is fully transparent to application, we can activate or deactivate it, the application will continue to work in the same way (they won't see any details). Also interesting because it can be transparent to end users. Furthermore, we can setup IPSec so that we can select different types of security for different types of users in the network. A practical application of IPSec could be building a secure link between two physical LANs for having only one logical LAN, and this is called a **tunnel** between the two LANs. IPSec is like a big protocol containing lot of small protocols implementing many different services such as :

- **Authentication Header (AH)** : it's the extension header for providing authentication.
- **Encapsulating Security Payload (ESP)** : it's the extension header for providing encryption.

We can use one of them or both. The more secure, the more demanding will be the application in terms of performance, so it's not so easy to perform the correct choice. Now, lets talk about the security features we can get from IPSec. We want to remind the basic philosophy of the protocol that is existing just over IP. This means whenever we have some application using TCP protocol, such request will be directed to IPSec and then the whole request will be incapsulated in an IP datagram. This means that the whole process of protecting IP layer is completely invisible to applications and to the users. IPSec provides the following features :

- **Access control** : it's the capability of preventing some non authorized usage of the resources. It means that every datagram that is going to some destination and such destination is offering services of whatever type, this can be analyzed so that it's possible to understand if the originator of the datagram is authorized to use the service.

- **Connectionless integrity** : we should remind that when we are just considering the traditional TCP/IP framework, we have datagrams sent over the network and the path of these datagrams is not perfectly defined, it may happen that a datagram is going faster than another datagram, the order in which they are received is not necessarily the correct one. It can also happen that some datagram not arrives at all to destination. In order to provide data integrity the TCP protocol needs a connection, it means there is a handshaking and while working the protocol is having a memory about the current state of the protocol. In general, we have another type of data integrity, the connectionless one, which means that the protocol doesn't needs to store the packets in memory.

- **Data origin authentication** : it verifies the identity of the claimed source of data, in order to be robust against replay attacks.

- **Confidentiality (encryption)** : there is the encryption that can be setup.

- **Limited traffic flow confidentiality** : we can have some confidentiality about the traffic flow, depending on how we use the AH and ESP protocol.

Another important concept used by IPsec is the concept of **security association (SA)**. This is a critical concept in the protocol, it's a relationship between the sender and receiver, a sender is sending a datagram, the receiver is receiving it. A security association provides information about this type of transmission, sender sending datagrams to the receiver and in order to understand what type of security should be associated with such datagram, we can use features in different ways, security association is a **one way relationship**. It defines the features of sending datagrams from Alice to Bob, details about sending datagrams from Bob to Alice are defined on a different security association, this is very important. All SAs are stored in an official database called **Security Associations Database (SADB)**, which is well defined in official documents. A SA is identified by three important types of information : a **Security Parameter Index (SPI)**, which is a pointer that points at technical information about the security we are going to use for sending the datagrams, the IP address of the other party and the **Security Protocol Identifier** which indicates what type of protocol should be used for such sending (AH or ESP). We should understand from this that, if we need to combine security features by adopting both of them, we will need two security associations and then we will combine them. Also if we want to have a bidirectional traffic, we need two security associations. The choice in the case we want to encrypt and authenticate the information, can be represented has a list of possible algorithms for the encryption and authentication, and such choice is left to the administration. The protection for outgoing packets is determined by the SPI. It looks like a port number, enabling the receiving system to select the SA under which a received packet will be processed. We have a similar procedure for incoming packets, where IPsec gathers decryption and verification keys from SADB. Now we want to describe the parameters defining the characteristics of a SA :

- **Sequence number counter** : it's a 32-bit value used to generate the sequence number field in AH or ESP headers.
- **Sequence counter overflow** : it's a flag indicating whether overflow of the sequence number counter should generate an auditable event and prevent further transmission of packets on this SA.
- **Anti-replay window** : it's a mechanism used to avoid the possibility of having attacks.
- **AH information** : they are information about AH protocol such as authentication algorithm, employed keys and other parameters.
- **ESP information** : they are information about ESP protocol such as encryption and authentication algorithm, employed keys and other parameters.
- **Lifetime of SA** : it's a time interval after which a SA must be replaced with a new SA.
- **IPsec protocol mode** : they are information about the way we want to implement the protocol, tunnel or transport.
- **Path MTU** : it's important to know about the MTU. If we have big packets then IP is fragmenting packets, since there exists several versions of fragmentation attack, may be we don't want to allow packets to be fragmented. A fragmentation attack has the goal of implementing a deny of service. When we send information about a fragmented

datagram, to implement that the IP protocol offers some tools for handling simple fragments, for example it uses some flags in the IP header informing that the datagram is a fragment, there are some offset showed in the IP header in order that the receiver is able to reconstruct the original datagram. A fragmentation attack is based on the fact that information provided for describing the current fragment are failing, with wrong information, wrong offset so that the algorithm collecting all fragments will crash.

There is another database, not only the database storing the security association, which is called the **security policy database**. It contains the first information about the fact that for some datagram we want IPsec protection or to bypass it. So the database contains entries for making differences in the packets defining the current traffic. Each database is associated with a set of IP numbers and a set of upper layer protocols, which are called **selectors**. Each entry in the database is pointing to a SA for that type of traffic, and we do not expect to have a 1-to-1 association between entry in this database and SA, because we can reuse some SA for different needs. Such selectors are used for implementing what is called **filter**, so we can analyze the traffic in order to map it into a particular SA. In order to realize this kind of filter the process is like this one : compare the values we are reading in the several fields in the packet against the SPD to find a matching SPD entry, which will point to zero or more SA. Once determined the SA (if any) for this packet, we need to understand the SPI and then we need to execute what is requested by the policy for the current datagram. We can use AH or ESP in one of the following two modes :

- **Transport mode** : in transport mode we need an extra header, instead in tunnel mode we encapsulate the whole datagram. This mode is in many cases preferred for a host-to-host communication. Only the payload of IP packet is encrypted and/or authenticated. However, there could be some issues in the case we are using the NAT because otherwise this will invalidate the hash value.

- **Tunnel mode** : it's the classical mode that is used for implementing a VPN. VPN is typically requesting the tunnel mode, most secure way. In the transport mode we can imagine that we have some original datagram to send, having a header and a payload. In the transport mode we need to add an additional header associated with IPsec. The router doesn't know that it's a datagram using IPsec, they will process it like a normal datagram. When we add this extra header, our goal is to protect the original datagram. This is the classical solution that is chosen when employing IPsec from end to end. With this mode we encapsulate every datagram in a new one. We understand that in this way we are hiding who is the destination of the packet, and using encryption, we hide also the port number. However, this approach is heavier since we encapsulate every single packet in another one, this is requesting some extra processing. We can also have issues using this approach because the original size is increasing, leading to a possible packet fragmentation.

In the AH with transport mode we will authenticate the payload and some portion of IP header that are meant to be never changed. In the case of the tunnel mode for AH we are making authentication over the whole original datagram plus selected portion of outer IP header and outer IPv6 extension headers. In the case of ESP without authentication we are encrypting the payload of the original IP packet, in the case we use the tunnel mode we encrypt not only the payload but also the original header of the original IP packet, becoming the new payload of the new datagram. In the case of ESP with authentication we are encrypting of course the IP payload and we authenticate the IP payload but not the header. In fact, AH authenticates also a small part of the header, ESP only the payload. In ESP with authentication in tunnel mode we are able to authenticate all the original packet because is the payload of a packet. So, the tunnel mode provides more security with respect to transport mode. However, this will affect the performance of the system.

Now we want to talk about the combination of security associations, since every SA can implement either AH or ESP. If we want to implement both, we need to combine two security associations, the result is called **security bundle**. We have two types of security bundles, he ones built using the transport adjacency and the ones obtained using the iterated tunneling, when we use two security associations over the same packet we have to process it and then the result according to the second security association. It's interesting to ask, i want to use AH and ESP, what i use first ? The simplest case is the one where we are using ESP with authentication, only one security association. It's easy, if we are using ESP with authentication we can setup a transport or tunnel mode. The other possible choice is to use the so called **transport adjacency**, we are assuming that we want authentication after encryption. Now, by using two security associations, we decide to bundle them in transport mode, so the inner security association is ESP.

ESP is providing encryption, in this case we don't ask for authentication, we will get it from the other SA. Then we protect the result of this encryption by using the AH protocol, providing authentication on the ciphertext. This means that the encryption is applied to the IP payload and the resulting packet is the IP header is the IP header and the ESP packet. AH is then applied in transport mode, so that authentication covers the ESP and the original IP header, except for mutable fields. We have talked about encryption first, what about authentication first ? If we want to do authentication before encryption, it's interesting for two reasons : authentication data if we authenticate first, then they will by protected by encryption. The second point is that if we want to verify the authentication of a message like plaintext, we need to re-encrypt the data. When we combine the two SAs we just have to decide the first and the second. Does it make sense to build bundles based on transport mode containing three security associations ? It doesn't make any sense, because we have only two protocols. Now, the original proposal of the IPsec protocol is considering 4 possible scenario that should be supported by the protocol :

- **Case 1** : it represents a host to host security association, starting from a host belonging to some intranet, the intranet is going outside by using some gateway, to the internet, then you may enter some other intranet with another gateway reaching another host. Here we can define AH in transport mode, ESP in transport mode, ESP followed by AH or combine them in a more complex as we want providing support for authentication, encryption, authentication before encryption, authentication after encryption.

- **Case 2** : it explicitly represents the VPN between a gateway of an intranet and a gateway of another intranet. The tunnel could supports AH, ESP or ESP with authentication. Typically we don't want here iterated tunnel, since IPsec services apply to the entire inner packet.

- **Case 3** : it represents a host to host security using the combination of multiple security associations, and inside this tunnel we will delivers secured packets, secured by other SAs.

- **Case 4** : it represents the situation in which we are at home and we want to access our computer at the office, our host at home is open to the internet and opens the security gateway of the enterprise and then packets are enabled to go through the local internet. We want multiple SAs, we want a VPN protecting us from our computer to the gateway, then we want to deliver packets within this tunnel, these packets can be unprotected or maybe we want to protect them because we need extra confidentiality.

Key management IPsec includes a big part for key management. The key management is very important because requires the possibility of setting up a key, the two parts should agree on a session key to do all the jobs, all is based on that session key. Two different protocols are employed : **Oakley** is a variant of DH, implementing a DH key exchange and preventing some typical attacks such as MITM attacks. **ISAKMP** is the protocol for setting up SAs and key management. It defines procedures and packet formats to establish, negotiate modify and delete SAs. What is missing is the set of the previous two protocols known as **Internet Key Exchange**. This is the standard, union of ISAKMP and Oakley.

9 TLS

We are going to consider the contribution of TLS. First we start by saying the original protocol was SSL, during the time the protocol started from version 1, 2, 3, after going to SSL version 3 the new version was TLS version 1. It's very different from making security at level of IPsec. The differences are that we are introducing a layer between TPC and application. The purpose of this layer is to secure the communication from port to port. Instead, IPsec provides security for communication from host to host. In this way we are securing an application, because it's an application that requires the connection from port to port. To make secure an application connecting Alice and Bob typically we don't need to secure all communications between them, but we need only to make secure the application between Alice and Bob, this means that we will prefer in most cases TLS against IPsec. Another difference is that IPsec is transparent to the applications. Instead, TLS isn't visible, because in this case we are changing the way how the applications are being interfaced to TCP. This is a cost, because we need to do some actions on the application to inform it that should make the request not to TCP directly but to another layer that is working over TCP. Typically we use TLS in a normal interaction to a web server by our web browser.Actually, we can use TLS in other applications. TLS provides authentication. When connecting to a website that is TLS enabled our browser is authenticating the other party, the website, our web browser will be sure about the identity of the web server. This is one features of TLS, this is a **one way authentication**. What is typically not known by most users is the fact that TLS can be used for implementing mutual authentication, typically the server will offers a digital certificate to the browser for proving its identity. What we could do is doing the same at browser side, installing a digital certificate on our browser proving our identity. This saves time for users, just connect and both parties are immediately authenticated. When making a TLS connection we have three phases :

1. Peer starts a conversation to understand what algorithm they can both support. They agree in this way on some algorithms to be used, encryption algorithm or hash functions and so on.

2. Then they do key exchange authentication.

3. After that they can start to run the normal communication offering encryption based on this symmetric ciphers and messages are all authenticated.

To be more precise, at the beginning client and server started talking about what suite of ciphers they know. So the client is saying ok i know these algorithms, the server is having a list of choices and from the known algorithms, it chooses the best one matching in the ordered list. Message authentication is typically implemented using again HMAC. In order to make the key exchange in most cases we have two prominent solutions, RSA or DH, or some variation of them. For having authentication we can use digital signatures based on RSA or DSA. For encryption we can use several algorithms implementing encryption, typically we want to use AES. The handshake of TLS protocol is the most complex part. If it fails, then the connection is not established. After the handshake we will have all the parameters set and we can start the secure conversation between the two parties and use them for some time, for the time we need. According to the original design of SSL, this was maintained in the subsequent versions, the protocol was based actually on two different layers of protocol. We have IP and TCP and then the SSL material. If we want to analyze the details of TLS we need to consider 4 protocols, the handshaking, a small protocol changing from the handshaking to the current session, a protocol for implementing a system of alerts for sending messages in case of failures and the standard secure protocol whose name is SSL record protocol, providing security offered by the SSL. The SSL record to be established needs to be activated by a handshaking that is demanding interaction between the two parties. Now, we need to understand two important concept : we have the concept of **SSL**

session and **SSL connection**. Session is a general framework, the result of a handshaking. The two parties establish several parameters and such parameters define a session, now within the session we can establish several connections, using the same parameters. The session defines a logical association between client and server, may be shared by multiple connections and is stateful. A connection is a communication link that is between the two parties, peer to peer, defined within a session and is stateful. In particular, a session is defined by several parameters, we have a session identifier, just a string of bits, a certificate, this is actually a description of the X.509 v3 certificate. This element can be null in the case the security is not based on the public key infrastructure. What compression method we are using, the specification of the cipher used, a master key, which is a shared secret between client and server. This shared secret is established as a result of the handshaking of course. Then there is a flag indicating whether the current session can be used for handshaking a new connection. Instead, the parameters that describes the state of a connection are the following : there is a nonce decided by the client and one decided by the server, then we have secret keys used for write operations originated by the server and write operations originated by the client. A write operation is sending a message, a read operation is getting a message. The two parties are able to use keys, one for encrypting and the other for decrypting. Other information needed is a seed, sequence numbers in order to contrast replay attacks and for being able to reconstruct the original sequence of packets. However, the most important protocol offering the service of guaranteeing the encryption, confidentiality,authentication and integrity of the messages is the **record protocol**. In order to get confidentiality, there are many encryption algorithms. We can get integrity using HMAC but with different padding. The record protocol works in the following way : when our application level protocol send some information from one party to the other party, application data are fragmented, compressed, HMAC is added, the result is encrypted and we get some data to which we add a header. The SSL header contains several fields such as content type, which defines what is the protocol asking the service to develop the protocol, major and minor version which indicates major and minor version of SSL in use respectively, and compressed length, which is the length of this portion of information that is transported by the protocol. What the payload is ? It's the information transported by the record protocol, we can have a normal payload like information that is having some meaning at http level or we can have other payload in the case where record protocol provides service to the handshake. The payload is organized in small portions, type of the message, length and content. We have another protocol called **change cipher specification**, it's just one byte that say ok we have done with our handshaking, now lets start the game by encrypting and authenticating messages. There is another protocol carrying information about possible problems, the **alert protocol**. Now, we want to remind some typical words that are employed while describing the TLS protocol, the concept of pending and current state. The pending state is what we are constructing while defining the details of cryptography we want to use, during the handshaking more information are added to define the pending state and at the end of the handshaking we have a pending state completely defined, we are ready to make this pending state the current real state. For what concern the alert protocol some bad facts may happen during the handshaking or during the normal operation of the protocol, we have two types of alert, **warning** or **fatal**. In the fatal case we are just closing the connection, in case of warning the connection continues but the higher level has to decide how to deal with this warning. The most interesting part of the protocol is the handshaking protocol. It's organized by the following phases :

1. **Hello** : the first step is the establishment of security capabilities. The client sends a list of possibilities, in order of preference. Server selects one, and inform the client of its choice. Parties also exchange random noise for use in key generation.

2. **Server authentication and key exchange** : server executes selected key exchange protocol (if needed). Server sends authentication information (e.g. X.509 certificate) to client.

3. **Client authentication and key exchange** : client executes selected key exchange protocol (mandatory). Client sends authentication information to Server (optional).

4. **Finish** : shared secret key is derived from pre-secrets exchange in 2) and 3) Change cipher protocol is activated. Summaries of handshake protocol are exchanged and checked by both parties.

We can define a session key between client and server using three possible variant of DH :

- **Fixed** : in order to do the fixed DH, the two parties are offering digital certificates and in the extra fields allowed in the standards X.509 there are some useful information like the public key for implementing DH. In most cases it happens that the server provide a digital certificate, otherwise we can't setup a TLS connection but the client can't offer a digital certificate, so in this case it's requested to do a client authentication or changing just the way we are doing the key exchange. This approach provide some useful security, since some information are stated and they can be derived from the digital certificates, so we cannot choose at any time a different set of parameters changing the prime number.

- **Ephemeral** : it's the case where the information for doing DH are exchanged by signing messages with RSA or DSS. They sign the messages to do the authentication of every single message. It's possible to authenticate both parties because we have public keys offered for doing the signature of messages. According to the ephemeral case, we have to consider that in theory we can consider this possibility more secure, because all parameters can be changed at any time and we can establish session keys belonging to a larger set. In the case of fixed DH actually the certification is showing the public key. In this case, we have the messages exchanged by the two parties that are signed, and the signature is made in some official way by using a certificate allowing the signature.

- **Anonymous** : this is the weak case, not allowed by the last version of TLS. It's the original one that is weak against the MITM.

In the second step of the handshake, we have the server authentication key exchange, the server sends information to the client offering a digital certificate may be offering a whole chain of digital certificates and information for the key exchange in order to construct a shared key. To do that the server sends some information depending on the type of construction that is chosen in the handshake. Other possible messages sent from the server to the client are the request to get a certificate and the information that the phase is completed. The message offering the certificate is the first message of the second phase and this is what we really need, because in all practical implementation of TLS today we want to be able to authenticate the server side, the client wants to see the digital certificate and of course this certificate is used for the ways we are setting up the key for the session and the only case where this is not used is the anonymous DH. In the case where fixed DH is used, the digital certificate contains the public DH parameters. Now, there are two possible cases, key exchange needed or not needed. It's not needed in the case where we use the fixed DH approach, we don't need to realize the subsequent steps for exchanging keys because the certificate already contains all information needed for constructing the key, or when we want to implement RSA key exchange, because in this case there is just a message being sent encrypting the session key. Instead, we need the key exchange in the case we use anonymous or ephemeral DH approach, since in the latter case the message includes the numbers defining the public key and a signature of such numbers. In the case of RSA key exchange it happens that the server is using RSA, but the public key for RSA is a public key for signature only. So, the server creates a temporary RSA public-private temporary pair in order to do encryption to send an encrypted information. Since the digital certificate supports RSA only for signatures, if we want to sign we need to encrypt, so we need to be able to generate a temporary pair. Now, in the fourth phase of the handshake, the two parties have completed the handshake and the client sends a message to the server with the information that he's ready to make current state and there is a similar message coming to the server, to start the encryption.

Another protocol that uses the record protocol is called **Heart Beating** protocol. It just implements a simple exchange of messages, "hey you, still there ? and the other one replies, "ok ,i'm still here". The implementation of this protocol in the open SSL software was bugged. Suppose we are sending to the other party a word. This bug was happened that the name of the attack was the header bleed, the attack is implemented in this way : if you are there and sends me a big number and a word of few letters, then, in the original implementation, the other party will reply by sending a high number of letters, this is a **buffer overflow** attack. In many cases information managed in the memory are cryptographical keys. This bug was allowing to expose them. There is another well known attack that is more complicated called **POODLE** attack. In order to make this attack, the client requests to the server to downgrade the conversion to SSL version 3 and after that the attacker can send some number of messages for getting partial information of subsequent encrypted messages. This is a way to break encryption for the TCP connection. In order to mitigate this kind of attack, we need to forbid to use this version of the protocol.

10 Firewalls

Firewalls have nothing to do with cryptography, they offer a different type of security. The security we can obtain is some added security, because they allow to implement policies like the simplest one which is called **blacklist-**

ing. For example, we don't want that people from our organization will able to connect to facebook, we can blacklist facebook and we can define a set of rules in our firewall that is preventing people from doing such connections. Basically the firewall is a device that is able to analyze datagrams that are going from the local network to outside and from outside to the local network. We describe policies implementing them by rules that are tools for enforcing the policies. We describe the rules in formal way and make such rules known to the firewall, so that it will use apply them in order to analyze every datagram that enter or leaves the internal network. After that the firewall will decide if the datagram is allowed to enter the network or not. If not the datagram will be dropped. Basically we can have three types of firewalls working at different levels, which allows to obtain different types of security. Of course the higher the level the greater is the power of the firewall to protect the network from bad traffic. Typically the firewall will allow a special area called **Demilitarized Zone (DMZ)**, where some actions that are forbidden in the internal network, here are allowed. Typically we have a gateway, that is the interface of such a network with the internet and we just have the firewall protecting the local network. In some cases, the firewall is implemented at gateway level, this is what happens when we consider our ADSL connection from the house to outside, we have a router containing a firewall. This is just the basic idea, we can have several variants, implementations and several ways to use a firewall. We can have two types of firewall, we can use a firewall for protecting a network or we can use a firewall for protecting one host, that is different. It means that we have a single host which implements a firewall, so that packets going to the host are analyzed against the rules in order to understand whether such packets are allowed. The firewall should be massively protected. We don't mean that the firewall is replacing cryptography, a firewall just adds extra security to what we get from cryptography. On the other hand, firewalls are useful but we cannot expect too much protection, in particular if some attack is originated in the local network, the packets will be not analyzed from the firewall, that will be useless in that cases. In many cases, the firewall is completely unaware of what is happening at application level, trojan or other unwanted software, by inspecting packets we can't recognize viruses and so on. In other words, a firewall doesn't protect us in a complete way. Actually we can have a firewall and decide whether the datagram is allowed to enter/exit the network or not. That is **packet filtering**. Instead, application level firewall acts like proxies, because they become the intermediates that will do the job for us, if the job is allowed. Typically is also true that the higher the level of the firewall the stronger the ability of the firewall will be. Naturally if a firewall is able to work at higher level is also able to work at lower levels. Application level firewalls are also able to do packet filtering. They are not really much used. The most used ones are the one based on packet filtering and application level filtering. In addition we can have our personal firewalls for protecting a single host and it can work at application level or at level of packet filtering. Another very important notion is the way of working, **stateless** or **stateful**. A firewall is working in a stateless way if every packet the firewall is analyzing is independent on other packets, so every packet is analyzed as a single packet. If the firewall maintains some memory about other packets the analysis is stateful because the firewall can have full or partial memory of other packets able to determine some relationship between them. However, making a firewall completely stateful is a very huge job, since of the huge amount of traffic typically managed by a firewall protecting a normal real private network. So, in the case we have some stateful firewall it's partially stateful, i.e. it maintains only partial information about the state of the connection. Typically packet filtering is based on IP numbers and TCP port numbers. The order of the rules that are checked in order to take a decision is meaningful. In fact, we need to define first the most specific rules and continue always with the more generic ones. Typically the first matching rules is applied. If there isn't any match, then the default rule is applied. Typically the decision of a matching rule is accept or reject. When we reject a packet we can use two different possibilities, just dropping the packet and the packet will disappear and the sender will not know about this action, or simply block the packet, but ICMP messages are sent back to the sender to inform that the delivery of the packet has failed. The design of a packet filtering is good because allow us to implement some basic policies, but it's not preventing attacks based on some specific application, it doesn't contains any mechanism for authentication. This means that we can receive a spoofed datagram, i.e. the IP number of the sender is fake, and in this case we cannot take any good decision. Another real problem is the fact that since from 10-15 years people are offered to use firewalls that are already configured, the firewall is having a default configuration. It look fine, but not really fine, because the default configuration is meaningless in our case. Our case is always a special case and we should be able to configure the firewall.

A fragmentation attack is based on the idea of splitting a packet into many smaller packets. We can do that for IP datagram and we have that the reconstruction phase fails just because information about the offset of the single fragments are wrong, leading to a crash of the target operating system. We can also perform a **syn flag attack**, it consists on the fact that a server is receiving many requests to open TCP connections, the connections remain half open, meaning that the TCP protocol is based on a three-way handshake. The client opens a connection, the server replies with an ack message and now it expects a message from the client, which it will never be sent. Many clients make the half open a connection, the server will have to use resources for storing information about those connections and after some time such information are releases because there is a timeout. If a new request arrives more frequently it happens that some time the failure at server side will accept new request for opening a TCP connection. Another limit of stateless filtering is the following, suppose that we have a mail server, the firewall for our local mail server contains some rules that they should allows the mail server to receive information from outside and making the server to reply in a correct way. Suppose also that a client needs to send the information to the server on port 25 and it's uses as a source port number, some high number. When the server replies to the client sends a packet from port 25 to some high numbered port. Now, if a packet from the mail server from port 25 is sent to some remote client with a high port number, should be allowed or not ? We don't know, to understand this we should say "ok, this is a reply to some already established connection". In fact, it doesn't make any sense that the mail server opens a new connection with some external client. To make a deeper analysis we need to introduce some small context. Indeed it's used what is called a **session filtering**, which is an approach where we analyze a packet within the context of a TCP session. We can say "ok, this is a packet replying to a request of opening a TCP connection". The context in many cases provides a very simple context like the TCP context, which provides user information enabling a deeper analysis making the firewall able to distinguish between two identical packets belonging to different contexts. In a session filtering framework, session filtering is introducing some memory, so is a stateful analysis.

10.1 Iptables

Iptables is a very well known software in the open source community, which allows to perform a stateful analysis of packets implementing the session filtering, managing NAT tables, and other thing like that. It works on some tables. A very important concept within iptables are the **chains**. A chain is a list of rules which can match a set of packets. Each rule specifies criteria for a packet and an associated target, namely what to do with a packet that matches the pattern. Iptables works with several types of tables such as the filter table, which is used to define the way the firewall is acting. There are other tables such as nat, mangle and raw table. Iptables allows also user defined chains in addition to its own built-in chains. If we consider the filter table we have the following chains :

- **INPUT** : it contains rules in order to decide what to do with an incoming packet, i.e. when a packet coming from outside is trying to reach some internal host.
- **FORWARD** : it contains rules in order to decide what do with a packet that traverse the firewall.
- **OUTPUT** : it contains rules in order to decide what to do with an outgoing packet, i.e. when a packet coming from some internal host is trying to reach some external host.

An important point is as a target we can set the name of a user defined chain. This allows us to implement a subroutine, a function which deals with special cases. We can extend the power of iptables by using extra modules such as the one for tracking connections.

11 Design choices

Now, the question is how should we design our network in order to use the firewall ? Where to place it ? The internet, protecting network, we have a router that is our gateway allowing external packets to reach our network. In most cases the gateway operating like a router is implementing packet filtering. This is not meant to be the real firewall, we can add some extra protection with our stateful firewall, so we have to decide how to manage the protection on our local network and a very typical approach is the one based on the so called **bastion host**. It is strongly protected, how ? We will be strongly protected by implementing physical measures for allowing access to the host only to authorized people. Also we can implement the so called **hardening** of the operating system. It means remove from the OS all the software we don't need. Imagine we have a host in ubuntu server distribution, we know that in it we have some standard software, compiler or some interpreter for python and other stuff. Having a compiler working in some host in principle is a vulnerability because if the attacker uploads a text file can run the compilation and getting an executable file, so just remove the compiler if we don't need that software in the host. Remove other unwanted software, all we don't really need. We have also some extra features, if we have decided to use a bastion host, in fact it will be the only directly linked from the internet. It's also good that it's working as a proxy server so can implement a firewall at application level. Other measures for protecting the network is checking the processes running, check file system, especially the part containing operating system and so on. If we are using a bastion host, the best solution is not to use one network adapter, but to have what is called a **dual bastion host**, i.e. a host with two network adapters. In this case there isn't a physical connection between internal and external networks. Of course if a bastion host is compromised is a dangerous, this is why we defines it as a special host to implement special measures. Another possible way to organize the network is using a **screened subnet** architecture, which uses two routers, one exterior and the other interior. In this case, only the screened subnet is visible to the external networks, i.e. the internal network is invisible.