Ca' Foscari University
Faculty of Computer Science

Cryptography

Afternotes

Author
Francesco Vivian

A.Y. 2020-2021

Lecture 1

Definition 1. Security: it generically refers to the possibility of "protecting" information, which is either stored in a computer system or transmitted on a network.

Whatever will be shown works in both situations.
To decide whether a computer system is "secure", you must first decide what secure means to you, then identify the threats you care about. Some threats are: cyberterrorism, denial of service, modified databases, virus, identity theft, stolen customer data, equipment theft, espionage.

There are different aspects to protect through security properties:

Property 1. Authenticity: an entity should be correctly identified.

Example 1. Some examples of authenticity:

- login process for authenticating a user (User Identification)
- a digital signature allows for authenticating the entity originating a message (Message Authentication)

Property 2. Confidentiality (secrecy): information should only be accessed (read) by authorized entities.

- Confidential information is not disclosed to unauthorized individuals (Data confidentiality)
- Individuals control what information related to them may be collected and stored and by whom that information may be disclosed (Privacy)

Example 2. Some examples of confidentiality:

- the person that accesses a database should be authorized to access the data
- personal privacy, my private data should be protected while browsing the web

The "access control" for confidentiality:

- use the "need to know" basis for data access. How do we know who needs what data? (Approach: access control specifies who can access what). How do we know a user is the person she claims to be? We need her identity and we need to verify this identity (Approach: identification and authentication).

- Analogously, the "need to access/use" is the basis for access to physical assets (access to a computer room, use of a desktop)

Confidentiality is difficult to ensure and easy to assess in terms of success: it is binary in nature (Yes/No).

Property 3. Integrity: information should only be modified by authorized entities.

- information and programs are changed only in a specified and authorized manner (Data integrity)
- a system performs its intended function, free from unauthorized manipulation (System integrity)

Example 3. We should not alter bank accounts and IoT device firmware.

If we don't have integrity, we also don't have confidentiality (with integrity I want only authorized users to be able to modify information, with confidentiality I want only authorized users to be able to see information, modifying includes seeing). Integrity is more difficult to to measure than confidentiality, it is non binary (it has degrees of integrity) and it is content-dependent (it means different things in different context)

Example 4. A quotation from a politician, we can preserve the quotation (data integrity) but mis-attribute (origin integrity), like *Y said that* instead of *X said that*.

Property 4. Availability: information should be available/usable fastly by authorized users.

Example 5. It is important to guarantee reliability and safety. Apart from attacks, availability might be loss in case of faults (we need to use fault-tolerant techniques). We need availability in case of a remote surgery for example (good QoS).

We can say that an asset (a resource) is available if:

- it provides a timely request response
- it provides fair allocation of resources (no starvation)
- it is fault tolerant (no total breakdown)
- it is easy to use in the intended way
- it provides controlled concurrency (concurrency control, deadlock control, ..)

Property 5. Non-repudiation: an entity should not be able to deny an event.

Example 6. Having sent/received a message. This property is crucial for e-commerce, where "contracts" should not be denied by parties.

Other properties that are not addressed in detail are: fairness of contract signing, privacy, anonimity and unlinkability, accountability, ..

Typical attacks

We will now see some typical attacks. We assume that information is flowing from a source to a destination (e.g.: reading data is a flow from the data container to a user, writing is a flow from a user to the file system).

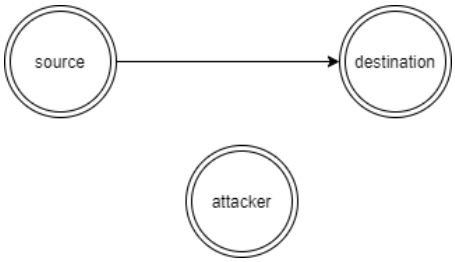

Figure 1: Expected information flow

Malicious users might try to subvert the properties previously mentioned in many different ways. We will now give a general classification depending on how an attacker might interfere on the expected flow of information (Figure 1).

Definition 2. Interruption: the attacker stops the flow of information (Figure 2). The attacker interrupts a service, it breaks system integrity and availability.

Some examples of interruption:

Example 7.

- the destruction of a part of the hardware
- canceling of programs or data files
- the destruction of a network link
- a denial of service (DoS) that makes the system/network unusable

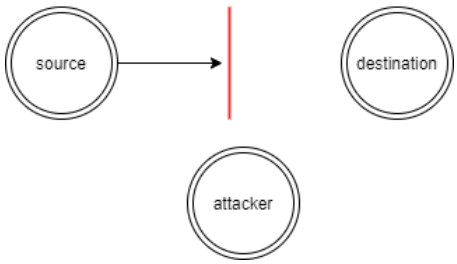

Figure 2: The attacker interrupts the flow of information

Definition 3. Eavesdropping (interception): the attacker gets unauthorized access to the information (depicted as an additional flow towards the attacker).

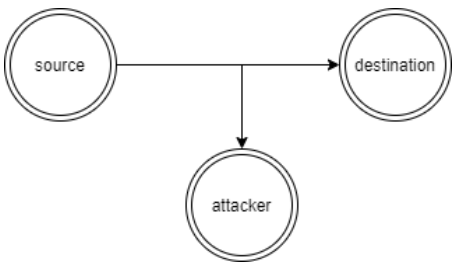

Figure 3: The attacker intercepts information

Interception is an attack to confidentiality, these attacks are hard to detect.

Example 8.

- unauthorized copies of files or programs;
- interception of data flowing in the network (a credit card number).

Interception attacks are hard to detect because source and destination don't notice any change (differently from interruption, where destination don't receive the flow).

Definition 4. Modification: the attacker intercepts the information and make unauthorized modification.

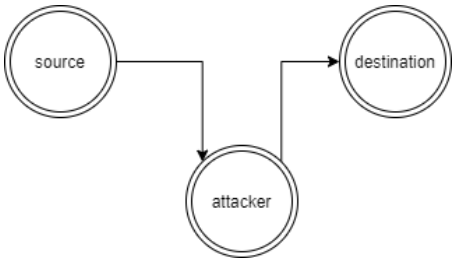

Figure 4: The attacker modifies information

In this case destination might notice that something is wrong.

Example 9.

- unauthorized change of values (e.g.: of a database);
- unauthorized change of a program;
- unauthorized change of data flowing in a network;
- A redirects S's bank transfer to herself (either in the browser or in the network, man in the middle).

Definition 5. Forging (falsification): the attacker inserts new information in the system (usually related to impersonation since the attacker lets the destination believe the information is coming from the honest source).

Forging is an attack to *authenticity, accountability* and *integrity*.

Example 10.

- addition of messages in the network;
- addition of a record in the database.

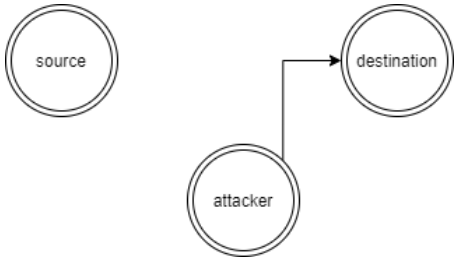

Figure 5: The attacker forges new information

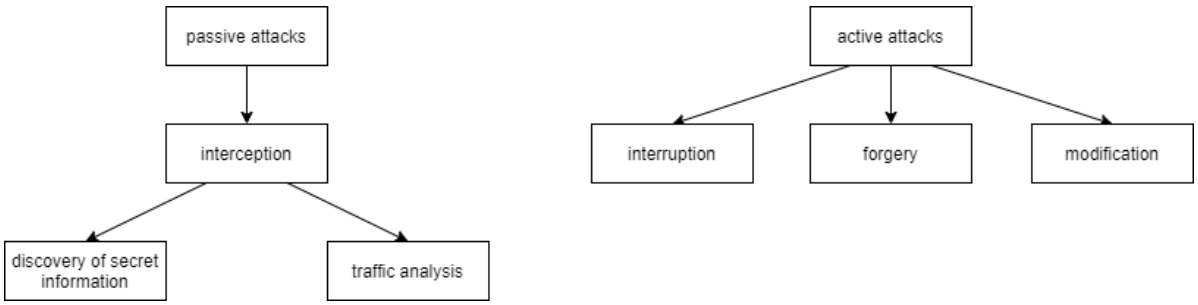

Figure 6: Classification of different types of attacks

Example 11. Suppose a bank B is using the following simple protocol to allow a bank transfer from user Alice (A):

$$\texttt{A} \rightarrow \texttt{B: sign\_A("please pay Bob 1000€")}$$

`sign_A` is some "signature" mechanism to ensure that the message really comes from Alice (thus the attacker cannot generate valid signed messages from Alice).

Let's suppose Bob is the attacker, he intercepts the whole message and repeats it as many times as he want, without modifying it. This attack (called replay) consists of an interception plus forging (in this case the message is just re-sent as it is). Bob obtains many bank transfers by just re-sending message M.

Example 12. Program modification: It is an attack to confidentiality, Bob modifies a program that is used by Alice, such a program apparently works normally, however it changes (e.g. the access rules of the users that are executing it, in this case it is called *Trojan horse*). Bob waits that Alice uses the program and copies all the files of Alice in his home directory.

Cryptography

The term *cryptography* comes from the greek and means "hidden writing". It is a way to protect the information when the environment is insecure. For example it is used when the information is sent on a network such as internet or when the system does not support sufficient protection mechanisms.

Definition 6. Encryption: a *plaintext* (message) is transformed using some rules (encryption algorithm) in a ciphertext.

Definition 7. Decryption: the plaintext is reconstructed starting from a ciphertext.

The decryption has to be simple for the receiver and unfeasible for an attacker. The information is encrypted in the source and travels to the destination where it will be decrypted. In order to do this there are two possible solutions:

1. only the sender (Alice) and the receiver (Bob) know the encryption algorithm. If the attacker, by looking at the flow of information, is able to guess which algorithm is used, then he will be able to decrypt all the messages sent;
2. the encryption algorithm is public and Alice and Bob share some information (the encryption key) non accessible by the attacker. If the attacker doesn't have the encryption key, it is unfeasible to decrypt the messages.

The second solution is better because it is simpler to distribute only one key and if there is an attack it is easier to change key instead of a whole algorithm.

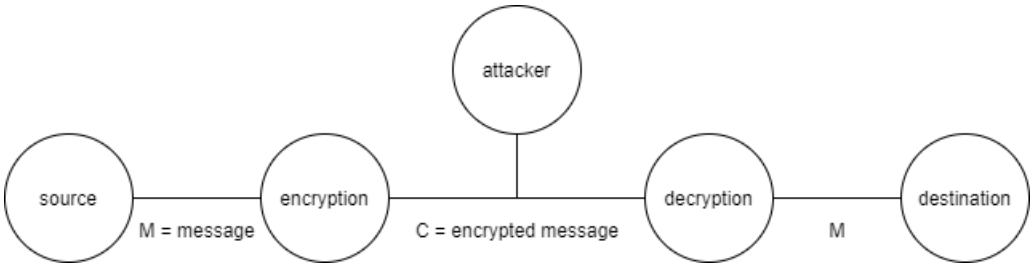

Figure 7: Encryption with a shared key

We want to build a secure channel to be able to exchange the encryption key.

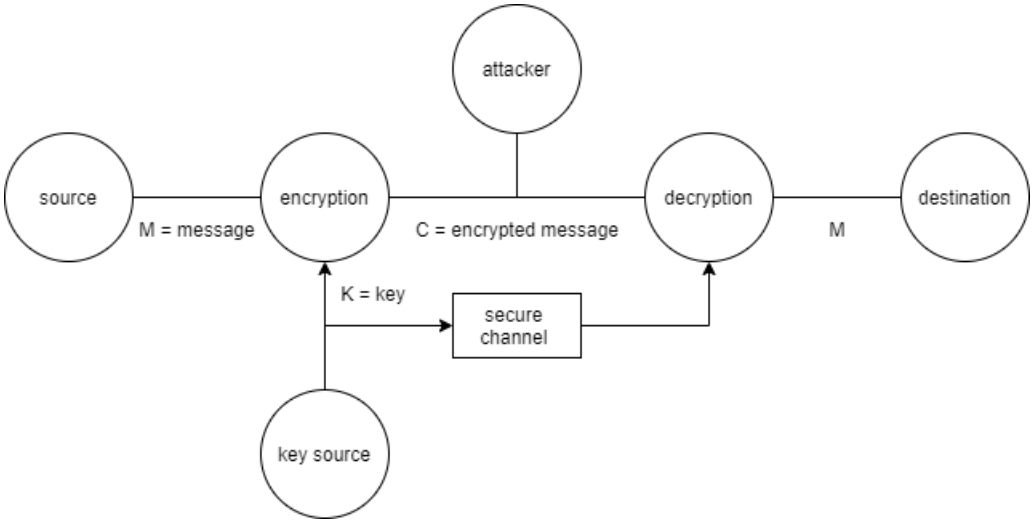

Figure 8: Encryption with a shared key and the secure channel

This is called *symmetric key cipher* (symmetric because source and destination use the same key).

One of the first encryption algorithms was used by Julius Caesar. In the Caesar Cipher all the letters are permuted using a certain rule (each letter is substituted by the one 3 positions ahead in the alphabet)

$$\begin{aligned} \mathtt{A} &\rightarrow \mathtt{D} \\ \mathtt{B} &\rightarrow \mathtt{E} \\ \mathtt{C} &\rightarrow \mathtt{F} \\ &\ldots \\ \mathtt{Z} &\rightarrow \mathtt{C} \end{aligned}$$

In this case the algorithm is the Caesar Cipher and the key is 3.

Lecture 2

The idea behind this course is to build step-by-step a system that is stronger and stronger until we'will arrive to systems that are used in practice.

Defining a cipher means to define an encryption algorithm and a decryption algorithm. A **cryptosystem** (or **cipher**) can be defined as a quintuple `(P,C,K,E,D)` where:

- `P` is the set of plaintexts (i.e. all the Italian words);
- `C` is the set of ciphertexts;
- `K` is the set of keys (we can have more than one key);
- `E:K×P→C` is the encryption function;
- `D:K×C→P` is the decryption function.

Let `x∈P, y∈C, k∈K`, we will write E_k(x) and D_k(y) to denote E(k,x) and D(k,y), the encryption and the decryption under the key k of x and y respectively.

We require two properties for each cipher that uses a shared key:

Property 6. $D_k(E_k(x)) = x$, decrypting a ciphertext with the right key gives the original plaintext.

Property 7. computing k or x given a ciphertext is infeasible, so complex that cannot be done in a reasonable time.

All the ciphers we will discuss have the first property, only "secure" ciphers have the second one (Caesar cipher doesn't). If someone, one day, will prove that P=NP, then most of the ciphers won't be secure anymore.

Example 13. Referring to the Caesar cipher, we can define the encryption function as "the letter three positions ahead (of our letter x) in the alphabet" or `(x+3) mod26` and the decryption function as "the letter three position behind (of the letter to be decrypted)" or `(x-3) mod26`. Our key k=3.

If we find the message `"BHV BRX PDGH LW"` and we know that it is encrypted with the Caesar cipher we can easily decrypt it into `"YES YOU MADE IT"` just going back 3 positions. We can notice that we have two "B" and they correspond to the same decrypted letter "Y", in monoalphabetic ciphers this is a strong weakness.

Proof of the first property applied to the Caesar cipher. We have to prove that:

$$\begin{aligned} x &= D_k(E_k(x)) && (1)\\ &= ((x+3) \bmod 26 - 3) \bmod 26 && (2)\\ &= ((x+3) - 3) \bmod 26 && (3)\\ &= x \bmod 26 && (4)\\ &= x && (5) \end{aligned}$$

Since we have the modules repeated twice we can keep only the external one. ■

To prove this property we can apply this reasoning to every cipher.

Definition 8. Kerckhoffs' principle: a cipher should remain secure even if the algorithm becomes public.

Kerckhoffs rules (1883):

- The system should be, if not theoretically unbreakable, unbreakable in practice;
- The design of a system should not require secrecy, and compromise of the system should not inconvenience the correspondents (Kerckhoffs' principle);
- the key should be memorable without notes and should be easily changeable;
- the cryptograms should be transmittable by telegraph;
- the apparatus or documents should be portable and operable by a single person;
- the system should be easy, neither requiring knowledge of a long list of rules nor involving mental strain.

The Caesar cipher is clearly insecure (it will be proved later) since once the cipher has been broken any previous exchanged message is also broken (as the cipher works the same way), the key should be changed and it is assumed to be the only secret.

Shift cipher

We can extend the Caesar cipher to a shift cipher with a generic key k, we can choose any key in the range $0 \leq k \leq 25$. For simplicity we will consider letters as numbers (`A=0, B=1, .., Z=25`), this means that $P = C = K = Z_{26}$ (Z_{26} is for all the integers between 0 and 25). For the encryption and decryption function we have:

$$\begin{aligned} \mathrm{E}_k(x) &= (x+k) \bmod 26 \\ \mathrm{D}_k(y) &= (y-k) \bmod 26 \end{aligned}$$

The Caesar cipher is a subcase of a shift cipher with k=3. In a shift cipher is useless to have k=0 because we would end up with equals plaintexts and ciphertexts.

Example 14. Considering k=10, it gives the following substitution:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
          ↓
KLMNOPQRSTUVWXYZABCDEFGHIJ
```

Proof of the first property applied to a shift cipher. We have to prove that:

$$\begin{aligned} x &= D_k(E_k(x)) && (1)\\ &= D_k((x+k) \bmod 26) && (2)\\ &= ((x+k) \bmod 26 - k) \bmod 26 && (3)\\ &= ((x+k) - k) \bmod 26 && (4)\\ &= x \bmod 26 && (5)\\ &= x && (6) \end{aligned}$$

■

Note that Z_{26} is a group under the addition (but not under the multiplication).

Definition 9. A **group** $\langle G, * \rangle$ is a set G together with a (closed) binary operation * on G such that:

- the operator is associative $((x * y) * z = x * (y * z)$ for all x,y,z in $\langle G, * \rangle)$;
- there is an element e∈G such that $a * e = e * a = a$ for all a∈G. Such an element is the identity element;
- for every a∈G, there is an element b∈G such that $a * b = e$. This b is said to be the inverse of a with respect to *. The inverse of a is sometimes denoted as a^{-1}.

The set $\langle Z, + \rangle$, which is the set of integers under addition, forms a group:

- addition is associative $((x + y) + z = x + (y + z)$ for all x,y,z in $\langle Z, + \rangle)$;
- the identity element is 0, since $0 + a = a + 0 = a$ for any a∈Z;
- the inverse of any a∈Z is -a.

A group that is commutative with an additive operator is said to be an abelian group.

The set $\langle Z, \cdot \rangle$, the set of all integers under multiplication, does not form a group. There is a multiplicative identity 1 but there is no multiplicative inverse for every element in Z.

From now on we will work with abelian groups.

Possible attack to shift ciphers

If I can attack a shift cipher, I can implicitly attack the Caesar cipher.

Example 15. If I am an attacker, I see the message `NGPPS` and I know that it is encrypted using a shift cipher but I don't know which key k is used I can try to get it by trial and error. I start with `k=1` and if decrypting the message I obtain something nonsense, I try with k=2 and so on. with k=4 I will reach that `NGPPS=HELLO`. Is it feasible? Yes because we only have 26 possible to keys to try. This type of attack is called **Brute force**.

In this case the problem with our encryption algorithm is the very small number of keys. Thus the second property doesn't hold (Kerckhoffs' principle: a cipher should remain secure even if the algorithm becomes public). The weakness is that we know that each letter is moved by the same distance.

Substitution cipher

A substitution cipher is a generalization to overcome the previous limitation: instead of moving all the letters by the same distance, now we use a generic permutation of the alphabet to map the letters. For example:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
          ↓
SWNAMLXCVJBUYKPDOQERIFHGZT
```

"HOME" becomes "CPYM". In this case the key is the complete permutation, otherwise the receiver is not able to decrypt our message. As for the previous ciphers, to decrypt we just apply the inverse substitution.
We have P=C=Z_{26} and K={p|p is a permutation of 0,..,25} with:

- E_k(x)=p(x);
- D_k(y)=p^{-1}(y).

Can we "Brute force" also this type of ciphers? No, we would have to try 26! keys, which is approximately $4 \times 10^{26} > 2^{88}$. This number of keys is very heavy to brute force, even with powerful parallel computers. We have to find something different to attack substitution ciphers. How can we do it?

It is a monoalphabetic cipher (it maps a letter to the very same letter), this preserves the statistics of the plaintext and makes it possible to reconstruct the key by observing the statistics in the ciphertext. For example, in Italian, almost all the words end with a vowel, and the vowels (a,e,i,o,u) are easy to identify as they are much more frequent than the other letters.

Let us assume a cryptoanalyst knows the used cipher (e.g. a monoalphabetic substitution cipher) and the language used in plaintext (e.g Italian), but he doesn't know the plaintext and the key.

Example 16. Given a ciphertext C, let us compute the frequency of letters, for example lette S appears 0 times and letter C appears 15 times. Is it possible that the cipher transforms A into S, and Z into C (A is never found and Z is found 15 times)? It is possible but very unlikely.

To compute the statistics for letters in Italian language it has been used a text of 14.998 letters, 1/3 from the book "Pinocchio" and 2/3 from the book "Il nome della rosa". The letters frequency are reported in the following table.

Letter	Absolute frequency	Percentage frequency
A	1714	0.114
B	160	0.011
C	637	0.042
D	566	0.038
E	1658	0.111
F	141	0.009
G	272	0.018
H	160	0.011
I	1563	0.104
L	966	0.064
M	436	0.029
N	966	0.064
O	1486	0.099
P	421	0.028
Q	85	0.006
R	978	0.065
S	771	0.051
T	1024	0.068
U	528	0.035
V	343	0.023
Z	123	0.008
Total	**14998**	**0.998**

Table 1: Letters statistics for Italian language.

The most used letter is A and the less used is Q. We can also build the graph in figure 9 to represent this statistic.

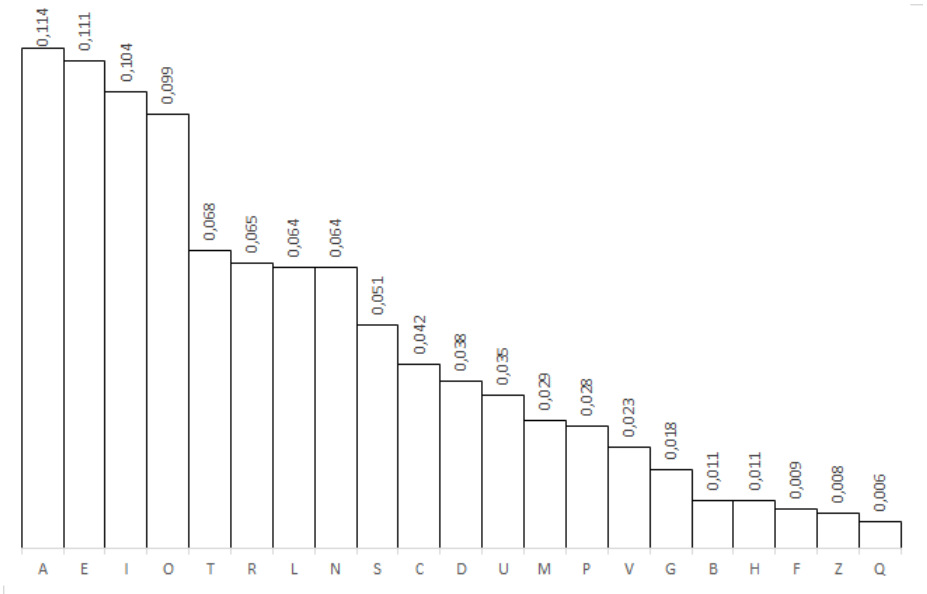

Figure 9: Percentage frequencies of letters in Italian language.

Knowing this, we can study the frequency of each letter in my ciphertext and map them using the frequency of my language.

- In Italian the frequency of letters I and L increases at the beginning of the sentences (articles "il", "lo", "la"), and the frequency of "A", "E", "I", "O" at the end of the words;
- there are words that may appear more frequently (e.g. the word "airplane" in a military message);
- there exist digraph and trigraph that are more frequent.

How can we decrypt a message?

- we order the letters of the ciphertext into decreasing frequencies;
- we substitute with letters in decreasing order as in the corresponding tables (depending on the language);
- there might be mistakes with letters that have the same frequency ("N" and "L" or "H" and "B" in Italian).

We need a long ciphertext in order to obtain reasonable frequencies. It is possible that at the first try we obtain some nonsense words but similar to some words of complete meaning, we can then change letters to correct these words. Repeating some times this step we would end up with the correct plaintext.

We have seen that we can easily break monoalphabetic ciphers applying this method, so the second property doesn't hold. Attacks to substitution ciphers are called **Statistical attacks**

Proof of the first property applied to a substitution cipher. We have to prove that:

$$\begin{aligned} x &= D_k(E_k(x)) && (1)\\ &= D_k(p(x)) && (2)\\ &= p^{-1}(p(x)) && (3)\\ &= x && (4)\\ & && (5) \end{aligned}$$

■

We have seen that monoalphabetic ciphers are prone to statistical attacks, since they preserve the statistical structure of the plaintext. A solution are the polyalphabetic ciphers in which the same symbol is not always mapped to the same encrypted symbol.

Polyalphabetic ciphers

An example of polyalphabetic cipher is the **Vigenére cipher** (XVI century). It works on "blocks" of m letters with a key of length m.

Example 17. The key is `FLUTE` (m=5). The plaintext is split into blocks of length 5 and the key `FLUTE` is repeated as necessary and used to encrypt each block.

THISISAVERYSECRETMESSAGE
+
FLUTEFLUTEFLUTEFLUTEFLUT
=
YSCLMXLPXVDDYVVJEGXWXLAX

Each letter of the ciphertext is given by the sum of the position of the letter in the plaintext plus the position of the letter in the key.

This type of ciphers works better than monoalphabetic ciphers because one letter is not always mapped to the same one unless they are at a distance that is multiple of m.

Formally, P=C=K=Z_{26}^m, where Z_{26}^m is $Z_{26}\times Z_{26}\times..\times Z_{26}$, m times.

- $E_{k1,..,km}(x_1,..,x_m) = (x_1+k_1,..,x_m+k_m)$ mod26;
- $D_{k1,..,km}(y_1,..,y_m) = (x_1-k_1,..,x_m-k_m)$ mod26.

If an attacker knows m, he has to try 26^m possible keys, if m is big enough, it is impossible to brute force it.

Lecture 3

Vigenére cipher cause an almost "flat" distribution of letters frequency, for this reason we would make many mistakes if we try do decrypt a ciphertext in the same way as we do for monoalphabetic ciphers.

As said in the last part of the previous lesson, if we want to brute force this cipher, (assuming we know m) we would have to try 26^m possible keys and that's infeasible, if we don't know m, it would we even worse. It is sufficient to choose m big enough to prevent brute force attacks, note that the number of keys grows exponentially with respect to the length.

Breaking Vigenére cipher

Even if the Vigenére cipher hides the statistic structure of the plaintext better than monoalphabetic ciphers, it still preserve most of it.

There are two famous methods to break this cipher, the first is due to Friedrick Kasiski (1863) and the second to Wolfe Friedman (1920). We will see the latter since it is more suitable to be mechanized. Both are based in **recover the length m of the key** and then **recover the key**.

The Friedman method uses statistical measures to recover the length m of the key. We consider the index of coincidence:

$$I_c(x) = \frac{\sum_{i=1}^{26} f_i(f_i - 1)}{n(n-1)} \approx \sum_{i=1}^{26} p_i^2 \tag{1}$$

where f_i is the frequency of the i-th letter in a text of length n, i.e., the number of times it occurs in such text and $p_i = f_i/n$ is the probability of the i-th letter. Intuitively, this measure gives the probability that two letters, chosen at random from the text, are the same. I compute this probability over all the letters.

Example 18. The IC of **"the index of coincidence"** is given by:
c(3*2) + d(2*1) + e(4*3) + f(1*0) + h(1*0) + i(3*2) + n(3*2) + o(2*1) + t(1*0) + x(1*0) = **34**
divided by n(n-1)=21*20 = **420**
which gives us an IC of 34/420 = **0.0809**

The IC of **"bmqvszfpjtcsswgwvjlio"** is given by:
b(1*0) + c(1*0) + f(1*0) + g(1*0) + i(1*0) + j(2*1) + l(1*0) + m(1*0) + o(1*0) + p(1*0) + q(1*0) + s(3*2) + t(1*0) + v(2*1) + w(2*1) + z(1*0) = **12**

divided by n(n-1)=21*20 = **420**
which gives us an IC of 12/420 = **0.0286**

Once I have the IC of my ciphertext, I have to compare it to the ICs of various languages to discover to which it corresponds.

The value of the index is maximum (value 1) for texts composed of just a single letter repeated n times. The value of the index is minimum (value $1/26 \approx 0.038$) for texts composed of letters chosen with uniform probability $1/26$.

The index of coincidence is thus a measure of how non uniformly letters are distributed in a text, each natural language has a characteristic index of coincidence, some examples:

$$\begin{aligned} \text{English} &\rightarrow 0.065 \\ \text{Russian} &\rightarrow 0.0529 \\ \text{German} &\rightarrow 0.0762 \\ \text{Spanish} &\rightarrow 0.0775 \end{aligned}$$

We can also use the Friedman method to find moro or polyalphabetic ciphers. We know that if we use frequencies analysis, if frequencies are flat, we have a polyalphabetic cipher, if we have peaks and valleys of frequencies, we have a monoalphabetic cipher. Considering the Friedman method, if the value of the index is minimum ≈ 0.038, we have a polyalphabetic cipher, if it is ≈ 0.065, we have a monoalphabetic cipher (same IC as English, just a permutation of letters).

With the Friedman method we can estimate m, the length of the key in a Vigenére cipher. The idea is to recover m by brute forcing, following this algorithm (in Python):

```
1  m=1
2  LIMIT = 0.06 #this is to check that ICs are above 0.06 and thus close to
                                        0.065 (assuming the text is in
                                        English)
3  found = False
4  while (not found):
5    sub = subciphers() #takes the m subciphertexts sub[m] obtained by
                                        selecting one letter every m
6    found = True
7    for i in range(0,m): #compute the IC of all subtexts
8      if IC(sub[i]) < LIMIT:
9        #if one of the IC is not as expected try to increase the length
10       found = False
11       m += 1
12       break
13 #survived the check, all ICs are above LIMIT
14 output (m)
```

It works because, once we reach the correct m, all the letters we are considering will be encrypted using the same key, "F" in the below example. So, computing the IC, we will obtain a value similar to the English IC value.

THISISAVERYSECRETMESSAGE
+
FLUTEFLUTEFLUTEFLUTEFLUT
=
YSCLMXLPXVDDYVVJEGXWXLAX

In order to obtain suitable results, we need to have a long enough ciphertext, otherwise we could not be able to succeed.

Now that we have m, we need to find the key. We already said that we cannot brute force it, it would be infeasible. What should we do?

- we divide the text into blocks of length m, as the length of the key (we just found it);
- we need to build new cryptograms with the first letter of each block, one with the second letter and so on;
- we analyse the new cryptograms as before and we find the shift in each position.

We are considering texts composed of letter at distance m from the first one, the second one, and so on. They have different shifts, we need to find the relative right shift.

The idea is to shift one subcipher until the mutual index of coincidence with the first subcipher becomes close to the one of the plaintext language, when this happens, we know that the applied shift is the relative shift between the two subciphers and , consequently, between the corresponding letters of the key.

The mutual index of coincidence is defined as:

$$MI_c(x, x') = \frac{\sum_{i=1}^{26} f_i f'_i}{nn'} = \sum_{i=1}^{26} p_i p'_i \tag{1}$$

It represents the probability that two letters taken from two texts x and x' are the same.

The following algorithm selects the relative shift that maximizes the mutual index of coincidence.

```
key = [] #empty list
for i in range(0,m): #for any letter of the key
  k = 0 #current relative shift
  mick = 0 #maximum index so far (we start with 0)
  for j in range(0,26): #for any possible relative shift
    #compute the mutual index of coincidence between the first subcipher
    #sub[0] and the i-th subcipher shifted by j
    mic = MIc(sub[0], shift(j,sub[i]))
    if mic > mick: #if it is the biggest so far
      k = j     #we remember the relative shift
      mick = mic  #.. and the maximum
  key.append(k)     #we append to the list the shift we have found
```

We repeat this for every letter of the key and we obtain the list of relative shifts, for example, if we obtain [0,4,6,3,9], it means that the second letter of the key is equal to the first plus 4, the third is equal to the first plus 6 and so on. But what is the first letter? The final step is to try all the possible 26 letter of the key, that gives us 26 possible keys.

Known-plaintext attacks

Until now we have considered attackers that only know the ciphertext y and try to find either the plaintext x or the key k. It is often the case that an attacker can guess part of the plaintext (e.g., the "standard" header of a message), if a message is split into blocks which are encrypted under the same key, it is reasonable to assume that an attacker can deduce part of the plaintext. For example if the attacker is trying to decrypt an email, he can guess the initial par that often starts with "Dear ...". If the attacker knows some plaintexts, this gives him the knowledge of some pairs (x,y) plaintext, ciphertext. Given this, the attacker should be able to decrypt other messages or to recover the key k (no matter which cipher is used).

The **Hill** cipher is a polyalphabetic cipher and it is a generalization of the Vigenére by introducing linear transformations of blocks of plaintext. We have P=C=Z_m^{26}, while K=$\{K|K$ is an invertible mod26 matrix m×m$\}$. The encryption and decryption are the following:

- $E_K(x_1,..,x_m) = (x_1,..,x_m)K$ mod26;
- $D_K(y_1,..,y_m) = (y_1,..,y_m)K^{-1}$ mod26.

Example 19. Let us assume M=(x_1,x_2)=(5,9) and K $=\begin{bmatrix}5 & 11\\ 8 & 3\end{bmatrix}$ (we will only consider 2×2 matrices for simplicity). Thus, $E_K(5,9) = (5,9) \times \begin{bmatrix}5 & 11\\ 8 & 3\end{bmatrix}$ mod26 $= (5 \times 5 + 9 \times 8, 5 \times 11 + 9 \times 3)$ mod26 $= (25 + 72, 55 + 27)$ mod26 $= (97, 82)$ mod 26 $= (19, 4)$

In order to decrypt a message, we need to compute the inverse of the matrix that is our key.

Example 20. We have (y_1,y_2) = (19,4) and K=$\begin{bmatrix}5 & 11\\ 8 & 3\end{bmatrix}$. To compute K^{-1}:

K^{-1} =$\det^{-1}$(K) $\begin{bmatrix}3 & -11\\ -8 & 5\end{bmatrix}$ mod26 = $\det^{-1}$(K)$\begin{bmatrix}3 & 15\\ 18 & 5\end{bmatrix}$ mod26.

We assume our matrix K is invertible. In the last step we changed the negative numbers to positive ones considering the mod26.

Now we can calculate det(K) $= (5 \times 3 - 11 \times 8)$mod26 $= (15 - 88)$mod26 = -73 mod26 = 5. How do we compute $\det^{-1}$(K)? To find the inverse mod26 of 5, we need to find a number in the interval [0,25] that multiplied by 5 mod26 gives 1. The number we are searching is 21, 5×21 mod26 = 105 mod26 = 1. Thus $\det^{-1}(K) = 21$.
Note that it is not always the case that the multiplicative inverse modulo exists, we will discuss this more in detail later on, introducing the public key cryptography and RSA.

Now we can solve $K^{-1} = \det^{-1}(K)\begin{bmatrix}3 & 15\\ 18 & 5\end{bmatrix}$ mod26= $21\begin{bmatrix}3 & 15\\ 18 & 5\end{bmatrix}$ mod26 $= \begin{bmatrix}63 & 315\\ 378 & 105\end{bmatrix}$ mod 26 $= \begin{bmatrix}11 & 3\\ 14 & 1\end{bmatrix}$. Thus $D_K(19,4) = (19,4)\begin{bmatrix}11 & 3\\ 14 & 1\end{bmatrix} = (19 \times 11 + 4 \times 14, 19 \times 3 + 4 \times 1)$ mod26 $= (265, 61)$ mod26 $= (5, 9)$.

Exercise 1. Encrypt and decrypt message (2,5) using a Hill cipher with $\begin{bmatrix} 5 & 11 \\ 8 & 3 \end{bmatrix}$

Encryption

$\mathrm{E}_K(2,5) = (2,5) \times \begin{bmatrix} 5 & 11 \\ 8 & 3 \end{bmatrix} \bmod 26 = (2\times5+5\times8, 2\times11+5\times3) \bmod 26 = (10+40, 22+15) \bmod 26 = (50, 37) \bmod 26 = (24, 11)$.

Decryption

$\mathrm{K}^{-1} = \det^{-1}(\mathrm{K}) \begin{bmatrix} 3 & -11 \\ -8 & 5 \end{bmatrix} \bmod 26 = \det^{-1}(\mathrm{K}) \begin{bmatrix} 3 & 15 \\ 18 & 5 \end{bmatrix} \bmod 26$.

$\det(\mathrm{K}) = (5 \times 3 - 11 \times 8) \bmod 26 = (15 - 88) \bmod 26 = -73 \bmod 26 = 5$.

$\det^{-1}(\mathrm{K}) = 21$, $5 \times 21 \bmod 26 = 105 \bmod 26 = 1$.

$\mathrm{K}^{-1} = \det^{-1}(\mathrm{K}) \begin{bmatrix} 3 & 15 \\ 18 & 5 \end{bmatrix} \bmod 26 = 21 \begin{bmatrix} 3 & 15 \\ 18 & 5 \end{bmatrix} \bmod 26 = \begin{bmatrix} 63 & 315 \\ 378 & 105 \end{bmatrix} \bmod 26 = \begin{bmatrix} 11 & 3 \\ 14 & 1 \end{bmatrix}$.

$\mathrm{D}_K(24,11) = (24,11) \begin{bmatrix} 11 & 3 \\ 14 & 1 \end{bmatrix} = (24 \times 11 + 11 \times 14, 24 \times 3 + 11 \times 1) \bmod 26 = (418, 83) \bmod 26 = (2, 5)$.