

Foundations of CyberSecurity

Computer Engineering@UniPi

Giovanni Enrico Loni

2 marzo 2024

Indice

0	Introduction	1
0.1	Outline of the course	1
1	Introduction to applied cryptography	3
1.1	Symmetric Encryption	3
1.1.1	The cipher	5
1.1.2	Attacks overview	5
1.2	Perfect cipher	6
1.2.1	A probabilistic view	6
1.2.2	Definition of a perfect cipher	7
1.2.3	Shannon's theorem	7
1.2.4	Perfect indistinguishability	7
1.3	One-time pad	8
1.3.1	Proof for XOR cipher	8
1.3.2	One-time pad has a perfect secrecy	8
1.3.3	One-time pad: pros and cons	8
1.3.4	One time pad malleability	9
1.3.5	Stream ciphers	9
1.3.6	Case study: 802.11 WEP	10

0 Introduction

0.1 Outline of the course

1. Applied cryptography
 - Definition **security**
 - Crypto-primitives: hoe they work and how to use them
 - Identification and authentication protocols
 - Case studies
 - Programming secure applications
2. Hands-on labs and challenges

1 Introduction to applied cryptography

Cryptography, or the art of secret writing, has been around for thousands of years, and get more and more important in our digital world, because of the need of secure communication and data protection in every aspect of our lives, such as online banking, e-commerce, e-mail, and social media, given the inevitable presence of malicious actors.

These malicious actors are often referred to as adversaries: they are **intelligent** users, with an *objective*, and lot of **resource** and **abilities**. This leads to an unfair competition between the adversaries and the defenders, also called the security engineers: the latter have to protect a system against all possible attacks, having in mind the usage and performance of the system, and this will be our role in this course.

Luckily, we don't have to reinvent the wheel: we can use tools and techniques that have been developed by cryptographers and security engineers over the years, and we talk about **applied cryptography** because of this very fact.

Remember these facts:

- Cryptography is a **useful** tool, but it is not a **silver bullet**.
- Cryptography is the basis of **security**, but it's not the solution to all security problems, and it's not reliable if not used properly.

1.1 Symmetric Encryption

Let's give some context to the usage of cryptography in the real world.

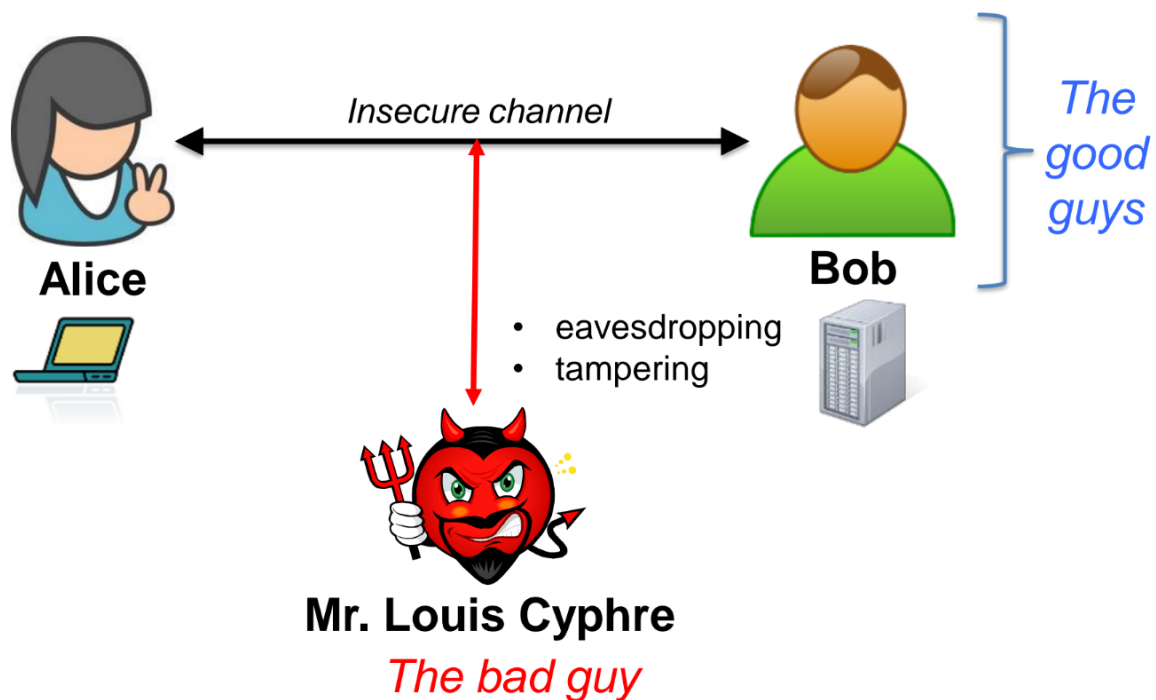


Figura 1.1: A simple scheme with our main actors

In the figure above, we have two main actors: Alice and Bob. They want to communicate securely, but they are afraid of the presence of an eavesdropper.

To allow a secure communication, we can use a model, defining the following elements:

- **Plaintext x** : the original message that Alice wants to send to Bob;
- **Ciphertext y** : the encrypted message that Alice sends to Bob;
- **Key k** : a shared secret between Alice and Bob, that is used to encrypt and decrypt the message;
- **Encryption function E** : a function that takes the plaintext and the key, and returns the ciphertext;
- **Decryption function D** : a function that takes the ciphertext and the key, and returns the plaintext;
- **Network**: the insecure communication channel between Alice and Bob.

Note that the encryption and decryption functions are publicly known, and the only secret is the key.

1.1.1 The cipher

We define as **cipher** the triple (K, P, C) of efficient algorithms (Gen, Enc, Dec) such that:

- Gen is a algorithm $Z^+ \rightarrow K$ that outputs a key;
- Enc is a algorithm $P \times K \rightarrow C$ that takes a key and a plaintext, and returns a ciphertext. An equivalent notations are $Enc(k, x)$, $E(k, x)$, or $E_k(x)$;
- Dec is a algorithm $C \times K \rightarrow P$ that takes a key and a ciphertext, and returns a plaintext. An equivalent notations are $Dec(k, y)$, $D(k, y)$, or $D_k(y)$.

Cipher **properties** are also important, and the main ones are:

- **Correctness:** for every key k and every plaintext p , it holds that $D(k, E(k, p)) = p$;
- **Security:** for every key $k \in K$, and for every encrypted $c \in C$, it's computationally infeasible to find both a plaintext p such that $E(k, p) = c$, without knowing k , and the key k itself.

1.1.1.1 Example: mono-alphabetic substitution

A simple example of a cipher is the **mono-alphabetic substitution**, where each letter of the alphabet is replaced by another letter, and the key is the mapping between the original and the substituted letters. The encryption algorithm is implemented by substituting each letter of the plaintext with the corresponding letter of the key, and the decryption algorithm is implemented by substituting each letter of the ciphertext with the corresponding letter of the inverse key. To break this cipher, we can think of a brute-force attack, where we try all the possible keys, but this is not feasible because of the large number of possible keys, which is $26! \approx 4 \times 10^{26}$. However, despite the large number of possible keys, the mono-alphabetic substitution is not secure, because of the frequency analysis attack, which is based on the fact that some letters are more frequent than others in a language, and this can be used to guess the key. From this example, we can see that the security of a cipher is not only based on the number of possible keys, but also on the structure of the cipher, and the security of the key: good ciphers should hide statistical properties of the plaintext, and their security mustn't rely on the dimensions of the key space.

1.1.2 Attacks overview

When we talk about the attack complexity, we refer to the number of operations that an adversary has to perform to break a cipher, and we can distinguish between three main types of attacks:

- **Ciphertext-only attack:** the adversary has access to a set of ciphertexts, and he has to find the plaintexts. This is the weakest type of attack, and it's often based on the frequency analysis, or on the knowledge of the structure of the plaintext;

- **Known-plaintext attack:** the adversary has access to a set of plaintext-ciphertext pairs, and he has to find the key. This is a stronger type of attack, and it's often based on the knowledge of the structure of the cipher;
- **Chosen-plaintext attack:** the adversary has access to a set of plaintexts, and he has to find the key. This is the strongest type of attack, and it's often based on the knowledge of the structure of the cipher, and on the ability to choose the plaintexts. We can easily see that if a cipher is secure against chosen-plaintext attacks, then it's also secure against known-plaintext attacks and ciphertext-only attacks.

1.1.2.1 Kerckhoffs' principle

The security of a cipher must rely only on the secrecy of the key, and not on the secrecy of the algorithm. This principle was formulated by Auguste Kerckhoffs in the 19th century, and it's still valid today, because it allows the public scrutiny of the algorithms, and it allows the development of secure ciphers, even if the adversary knows the algorithm. This because is **easier** to keep a small key secret, than a large algorithm secret: in case of a leak, changing the key is easier (and cheaper) than changing the algorithm.

1.1.2.2 Security trough obscurity

The opposite of Kerckhoffs' principle is the **security trough obscurity**, which is the reliance on the secrecy of the algorithm, and it's a bad practice, because it's not reliable, and it's not a good security measure. This is because the adversary can reverse-engineer the algorithm, and find the key, and because the algorithm can be leaked, and the security of the system is compromised. This has been proved over the year: GSM/A1 were disclosed by mistake, Enigma was broken by intelligence, and the DVD encryption was broken by reverse-engineering.

1.2 Perfect cipher

In the hypothesis such that attacker isn't able to recover both some plaintext and the key, a powerful intuition is, regardless of any *a priori* knowledge, the ciphertext shouldn't reveal any information about the plaintext.

1.2.1 A probabilistic view

Let a message M be a random variable, with a probability distribution P_M such that $P[M = \text{"attacktoday"}] = 0.7$ and $P[M = \text{"don't attack"}] = 0.3$ knew *a priori* by the attacker, and let $Gen()$

a probability distribution over the keys such that $P[K = \text{"key1"}] = P[k \leftarrow \text{Gen}()]$, with M and K independent. To generate the ciphertext, we can use the following algorithm:

1. choose a message m according to $P[M]$;
2. generate the key k according to $P[K]$;
3. compute the ciphertext $c = E(k, m)$.

1.2.2 Definition of a perfect cipher

It's clear that the ciphertext c is a random variable C , defined by the encryption function E , so we can formalize the concept of *information about plaintext* in terms of probability distributions. A cipher is said to be **perfect** if the ciphertext C is independent of the plaintext M : $P[M = m|C = c] = P[M = m]\forall m, c$. This means that the attacker can't learn anything about the plaintext by observing the ciphertext. If a cipher is perfect, it's said that it's **perfectly secure**, and the necessary conditions for a cipher to be perfectly secure are that the key must be chosen uniformly at random, and that the key space must be at least as large as the message space, as stated by Shannon's theorem.

1.2.3 Shannon's theorem

In a perfect cipher, the key space must be at least as large as the message space. Speaking on mathematical terms, $|K| \geq |M|$. We can prove this by contradiction:

1. suppose that $|K| < |M|$, which is the contradiction;
2. from the previous point, it must be true that $|C| \geq |M|$, because the cipher must be invertible;
3. combining the two previous points, we have that $|C| > |K|$;
4. now select $m \in M$ such that $P[M = m] > 0$, and $c \in C$ such that $E(k_i, m) = c \forall k_i \in K$;
5. because of point 4, must exists at least one c such that $c \neq c_i$;
6. therefore, $P[C = c|M = m] = 0$, that is a contradiction because the cipher is perfect, so holds that $P[M = m|C = c] = P[M = m] > 0$.

1.2.4 Perfect indistinguishability

An encryption scheme $\Pi = (G, E, D)$ over (K, M, C) is said to be **perfectly indistinguishable** if:

- $\forall m_1, m_2 \in M, |m_1| = |m_2|$;
- $k \leftarrow \text{Gen}()$ is chosen uniformly at random;
- $\forall c \in C, P[E(k, m_1) = c] = P[E(k, m_2) = c]$.

This means that the attacker can't distinguish between two plaintexts by observing the ciphertexts: it's also true that Π is perfectly indistinguishable if and only if it's perfectly secure.

1.3 One-time pad

The one-time pad is a perfectly secure cipher, and it's based on the following assumptions:

- let x be a t -bit message, i.e. $x \in \{0, 1\}^t$;
- let k be a t -bit key, i.e. $k \in \{0, 1\}^t$, where each bit is chosen uniformly at random;

The encryption function $E(k, x)$ is defined as the bitwise XOR between the message and the key, and the decryption function $D(k, y)$ is defined as the bitwise XOR between the ciphertext and the key.

1.3.1 Proof for XOR cipher

Let X a random variable over $\{0, 1\}^t$, and K a random variable over $\{0, 1\}^t$, with X and K independent. Then, define $Y = X \oplus K$, where \oplus is the bitwise XOR. We want to prove that Y is uniformly distributed over $\{0, 1\}^t$. The proof, for $t = 1$, is the following:

- $P[X = 0] = X_0, P[X = 1] = X_1$, such that $X_0 + X_1 = 1$;
- $P[K = 0] = K_0, P[K = 1] = K_1$, such that $K_0 + K_1 = 1$;
- $P[Y = 0] = P[X = 0, K = 0] + P[X = 1, K = 1] =$
 $= P[X = 0]P[K = 0] + P[X = 1]P[K = 1] =$
 $= X_0K_0 + X_1K_1 = X_0 \cdot 0.5 + X_1 \cdot 0.5 =$
 $= 0.5 \cdot (X_0 + X_1) = 0.5.$

1.3.2 One-time pad has a perfect secrecy

The proof for the one-time pad is the following:

1. from **Bayes' theorem**, we have that $P[M = m|C = c] = \frac{P[C=c|M=m]P[M=m]}{P[C=c]}$;
2. from **Total probability theorem**, we have that $P[C = c] =$
 $\sum_i^{|M|} P[C = c|M = m_i]P[M = m_i] = \sum_i^{|M|} P[K = c \oplus m_i]P[M = m_i] =$
 $= \sum_i^{|M|} \frac{1}{2^t} P[M = m_i] = \frac{1}{2^t};$
3. substituting the previous point in the first one, we have that $P[M = m|C = c] =$
 $P[K = c \oplus m] \frac{P[M=m]}{2^t} = 2^{-t} \frac{P[M=m]}{2^t} = P[M = m].$

1.3.3 One-time pad: pros and cons

The one-time pad has some pros:

- it's perfectly secure;

- it's simple and efficient;
- only one key maps a message m to a ciphertext c .

On the other hand, it has some cons:

- the key must be as long as the message, so it's almost impossible to use it in practice;
- keys must be used only once, so they must be shared in advance, and this is not always possible;
- a known-plaintext attack is possible, because the key can be recovered by XORing the plaintext and the ciphertext.

1.3.4 One time pad malleability

A crypto scheme is said to be **malleable** if it's possible to modify the ciphertext in such a way that the resulting plaintext is related to the original plaintext. The one-time pad is malleable, because if we XOR the ciphertext with a message, we obtain a new ciphertext that decrypts to a message related to the original message. A practical example is the following:

- suppose that Alice sends a message $c = p \oplus k$ to Bob, and Eve intercepts the message;
- Eve can modify the message by XORing it with another message m , and send the new message $c' = c \oplus r$ to Bob, where r is a *perturbation*;
- Bob now receives the message c' , and decrypts it to obtain $p' = c' \oplus k = c \oplus r \oplus k = p \oplus k \oplus r \oplus k = p \oplus r$.

The perturbation goes unnoticed, and can have a significant impact on the plaintext.

1.3.5 Stream ciphers

The one-time pad is a stream cipher, and it's based on the concept of a **keystream**: a sequence of bits that is used to encrypt the message. The keystream is generated by a **keystream generator**, which is a function that takes a key and a nonce, and returns the keystream. The keystream is then XORed with the message to obtain the ciphertext, and the same keystream is then XORed with the ciphertext to obtain the message. The keystream generator must be a **pseudo-random generator**, which is a function that takes a seed and returns a sequence of bits that is indistinguishable from a truly random sequence of bits. The keystream generator must also be **deterministic**, which means that the same seed must always produce the same sequence of bits.

When we refer to a **pseudo-random generator**, we mean:

a deterministic algorithm that, given a truly random binary sequence of length k , called the **seed**, produces a binary sequence of length n , called the **output**, such that the output is indistinguishable from a truly random sequence of length n .

If a pseudo-random generator has these two main properties:

- it's **computationally secure**, which means that it must be computationally infeasible to distinguish the output from a truly random sequence;
- it's **unpredictable**.

then is called **cryptographically secure pseudo-random generator**. Formally, we can say that given a sequence of bit $s_i, s_{i+1}, s_{i+2}, \dots, s_{i+n-1}$, it's computationally infeasible to predict s_{i+n} or, in other words, doesn't exist an algorithm that can predict the next bit of the sequence with a probability greater than 0.5.

1.3.6 Case study: 802.11 WEP

The Wired Equivalent Privacy (WEP) is a security protocol for wireless networks, and it's based on the RC4 stream cipher. Its structure is the following:

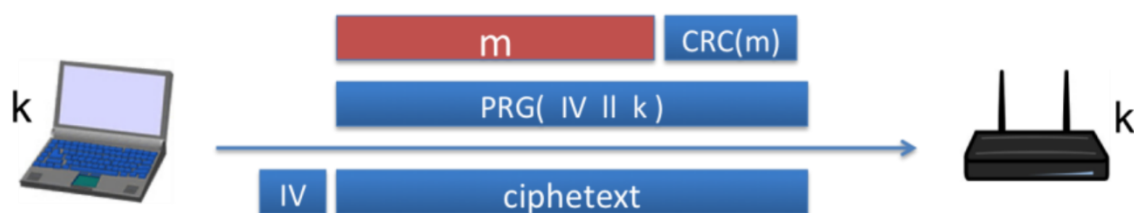


Figura 1.2: RC4 structure

The key length is fixed to 104 bits, and the IV length is fixed to 24 bits; it can be settled with a counter, or with a random number. The IV is concatenated with the key, and the result is used as the seed for the keystream generator. The keystream is then XORed with the message to obtain the ciphertext, and the same keystream is then XORed with the ciphertext to obtain the message. During the years, WEP has been proved to be insecure, because of the following reasons:

- the key is too short and can be used more than once, and this can lead to the recovery of the key. In particular, it's possible to recover the value of K in about 40'000 frames.
- the repetition of the IV after 2^{24} frames, and this can lead to the recovery of the keystream.
- **CRC**, used to check the integrity of the message and chosen for its execution speed, is not secure and can be easily forged.