Università degli Studi di Modena e Reggio Emilia

FACOLTÀ DI INGEGNERIA
Corso di Laurea Triennale in Ingegneria Informatica

Prova Finale

Non so ancora il titolo

Candidato:
Luca Lumetti
Matricola 118447

Relatore:
Prof. Maurizio Casoni

Correlatore:
Dott. Martin Klapez

Anno Accademico 2019-2020

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduction

Capitolo 2

Cryptography Overview

Cryptography is the study of using digital coding to secure data access. In other words to ensure that data can only be access by authorized entities.
To describe the problem that cryptography is trying to address we will use a specific jargon to represent entities: Alice and Bob.
Alice has a message that wants to share with Bob so they are both authorized to read the message. Cryptography becomes relevant when there's an adversary, or an attacker, that try to access the data sent from Alice without the legitimate authorization.
The *plaintext* is the message that Alice wants to send to Bob. The *ciphertext* and is the data that goes through the channel and one of the resources that an adversary can access. The process of converting the plaintext to the ciphertext is called *encryption* while the process to convert the ciphertext to the plaintext is called *decryption*. Encryption and decryption are defined by the *cryptographic scheme*, a set of algorithms that Alice and Bob decide to use before the actual communication, and one or more *keys*, that can be confidential only between the two entities or they can also be public depending on the type of scheme used.
Regardless of the type of scheme used, which will be discussed later in detail, there are three main features that a cryptographic scheme should have to be defined secure: *confidentiality*, *integrity* and *authentication*.

2.1 Confidentiality

Confidentiality assures that in a communication, an eavesdropper is unable to obtain any information about the messages exchanged or the key used to encrypt them. This means that the ciphertext should appear to the adversary as completely random garbage.

2.1.1 Perfectly Secret

We denote with $\mathcal{M}$, $\mathcal{K}$, $\mathcal{C}$ respectively the message's space, key's space and ciphertext's space, with $\mathsf{Pr}[M = m]$ the probability that a message m is sent and with $\mathsf{Pr}[C = c]$ the probability that the ciphertext is c.
We can define a cryptographic scheme to be perfectly secret if for every $m \in \mathcal{M}$, every $k \in \mathcal{K}$:

$$\mathsf{Pr}[M = m|C = c] = \mathsf{Pr}[M = m]$$

This means that the distribution over $\mathcal{M}$ is independent of the distribution over $\mathcal{C}$. To achieve this definition, the key's space $\mathcal{K}$ must be greater than the message's space $\mathcal{M}$. This can be impractical and inconvenient because perfect secrecy is defined against an adversary with unbounded computational power. We can relax this latter constraint to just be secure against polynomial-time algorithms.

2.1.2 Computationally Secret

Computational security is the aim of most modern cryptographic schemes. Modern encryption schemes can be broken given enough time and computation, nevertheless, the time required even for the most powerful supercomputer today built is in the order of hundreds of years.
From the previous definition of perfect secrecy we add two relaxations:

- Security is only preserved against efficient adversaries.
- Adversaries can potentially succeed with a negligible probability.

With the term efficient, we refer to an algorithm that can be carried out in *probabilistic polynomial time* (PPT). An algorithm A is said to run in polynomial time if there exists a polynomial $p(\cdot)$ such that for every $x \in 0, 1^*$, $A(x)$ terminates within at most $p(|x|)$. A probabilistic algorithm is one that has access to some randomness so its results depend on changes.
With negligible probability, we refer to a probability asymptotically smaller than the inverse of every polynomial $p(\cdot)$. So a function $f(\cdot)$ is $\mathsf{negligible}$ (typically denoted with negl) if for every polynomial $p(\cdot)$ there exists an N such that for all integers $n > N$ it holds that $f(n) < \frac{1}{p(n)}$.

2.2 Integrity

Integrity assures to the receiver that a message is not corrupted or that an adversary modifies and relays it (for example in a man-in-the-middle attack).

The decryption of the message is not always needed to modify it but can be enough to have the ciphertext.
Hash functions are used to assure the integrity of a message.

2.2.1 Hash Functions

In general, hash functions are just functions that take arbitrary-length strings and compress them into shorter strings. A hash function is a pair $(\mathsf{Gen}, \mathsf{H})$ such that:

- Gen: is a randomized algorithm that takes as input a security parameter n and outputs a key s
- H: is a deterministic polynomial-time algorithm that takes as input a string $x \in \{0,1\}^*$ and a key s to outputs a string $\mathsf{H}^s(x) \in \{0,1\}^{l(n)}$ where l is a polynomial.

In practice, hash functions are unkeyed, or rather it is included in the function itself. As an example of use of hash functions, imagine that a user A want to send a message m to an user B and he also want to assure its integrity. After they both agree on the hash function to use, A send $(m, \mathsf{H}(m))$, then upon receiving the pair, B itself calculate $\mathsf{H}(m)$ and verify that it is the same it received from A. If they match, m can be considered intact.
The domain of H is unlimited, instead, its image is limited. For the pigeon-hole principles this means that there are infinite pairs of differents string x and x' such that $\mathsf{H}(x) = \mathsf{H}(x')$, this is also known as a *collision*.

2.2.2 Collision-resistant hash functions

Hash functions used in cryptography are also called collision-resistant hash function, to emphatize the importance to have the property that no polynomial-time adversary can find them in a reasonable time. There are 3 levels of security:

1. **Collision resistance:** is the most secure level and implies that, given the key s, is infeasible to find two different values x and x' such that $\mathsf{H}^s(x) = \mathsf{H}^s(x')$.
2. **Second preimage resistance:** implies that, given s and a string x, is infeasible for a polynomial-time algorithm to find a string x' such that it collide with x
3. **Preimage resistance:** implies that given the key s and an hash y, is infeasible for a polynomial-time algorithm to find a value x such that $\mathsf{H}^s(x) = y$.

Notice that every hash function that is collision resistant is second preimage resistant, also a second preimage resistant function is a preimage resistant function.

2.2.3 Birthday Attack

The probability of finding a collision by guessing two random string is inversely proportional to the number of bit outputted by the hash function. For example, SHA-256 has a 256-bit hash output resulting in 2^{256} possible results.
Now assume we have are given an hash function $\mathsf{H}^s : 0,1^* \to 0,1^{l(n)}$. Then we choose at random q distinct strins $x_1, ..., x_q$ and compute $y_i := \mathsf{H}^s(x_i)$, $\forall i \in 1, ..., q$. If $q \leq 2^{l(n)}$ the probability is:

$$1 - \prod_{i=1}^{q-1}(1 - \frac{i}{2^{l(n)}})$$

This problem has been extensively studied and is related to the so-colled birthday problem and an approximation of the above formula is:

$$\frac{q^2}{2 \cdot 2^{l(n)}}$$

If we want to know how much q are needed to have a probability at least of 50% :

$$\frac{q^2}{2 \cdot 2^{ln(n)}} = 0.5 \quad \Longrightarrow \quad q = \sqrt{2^{ln(n)}}$$

This mean that if the output length of a hash function is $l(n)$ bits then the birthday attack finds a collision in $\mathcal{O}(q) = \mathcal{O}(2^{l(n)})$ time.

2.3 Authentication

In a cryptographic scheme, authentication is needed to authenticate the entities involved in the comunication. In other words, authentication assure that messages received by Bob are for sure from Alice.
To ensure authentication hash functions are used and that's why, in general, authentication imply integrity (but the opposite is not true).

2.3.1 The Merkle-Damgård Transform

Even if we have defined hash functions as functions with an infinite domain, in practice they are first constructed to be *fixed-length*, that means their domain is finite,

then they are extended to cover the full domain $0, 1^*$.
This extension is made easy by the Merkle-Damgård transform which also preserve the collision resistant propriety.
We will denote the given fixed-length collision-resistant hash function (or *compression function*) by $\mathsf{Gen}_h, h)$ and use it to construct a general collision-resistant hash function (Gen, H) that maps inputs of any length to outputs of length $l(n)$.
Let $\mathsf{Gen}_h, h)$ be a fixed-length hash function with input length $2l(n)$ and output $l(n)$.
Construct a variable-length hash function (Gen, H) as follow:

- $\mathsf{Gen}(n)$: upon input n, run the key-generation algorithm Gen_h. $s \leftarrow \mathsf{Gen}_h$.
- $H^s(x)$: upon input key s and message $x \in 0, 1^*$, compute as follows:
  1. Pad x with zeroes until it's length is a multiple of $l(n)$. Let $L = |x|$ (length of the string) and let $B = \left\lceil \frac{L}{l(n)} \right\rceil$ (number of blocks of length $l(n)$).
  2. Define $z_0 := 0^{l(n)}$ and then $\forall i = 1, ..., B$, compute:

     $$z_i := h^s(z_{i-1}||x_i)$$

     where h^s is the given fixed-length hash function.
  3. Output $z = H^s(z_B||L)$.

Capitolo 3

Private Key Cryptography

With "private-key cryptography" we refer to schemes that use a single key to encrypt and decrypt a message, for this reason, they are also called "symmetric key schemes". These are the first scheme ever created and they are still widely used today. They are defined as follows:
A **private-key scheme** is a tuple $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ such that:

- $\mathsf{Gen}(\cdot)$**:** is a randomized polynomial algorithm that generates the key. It takes as input a security parameter n and outputs a key k that satisfies $|k| \geq n$.
  We will write this as $k \leftarrow \mathsf{Gen}(1^n)$.

- $\mathsf{Enc}(\cdot)$**:** is a probabilistic polynomial-time algorithm that encrypts the message (or other forms of information) to send. It takes as input a key k and a message m to outputs a ciphertext c. We will refer to the unencrypted message also as plaintext.
  We will write this as $c \leftarrow \mathsf{Enc}_k(m)$.

- $\mathsf{Dec}(\cdot)$**:** is a deterministic polynomial-time algorith that takes as input a ciphertext c and a key k, and outputs a plaintext m.
  We will write this as $m := \mathsf{Dec}_k(c)$

It's also required that for every n, every k and every m it holds that $m = \mathsf{Dec}_k(\mathsf{Enc}_k(m))$.

Capitolo 4

Conclusions

Bibliografia