Applied Cryptography

Edoardo Righi

January 8, 2020

Contents

I CRYPTOGRAPHY . . . 2
1 Introduction . . . 3
2 Attacks . . . 5
3 Shannon's theorem . . . 7
4 From Shannon to modern cryptography . . . 10
5 Stream ciphers . . . 12
6 Linear Feedback Shift Registers . . . 15
7 Cryptography in GSM: the A5 family . . . 17
8 Cryptography in Bluetooth: E0 and RC4 . . . 18
9 DES-3DES . . . 20
10 Advanced Encryption Standard (AES) . . . 21
11 Serpent & Present . . . 24
12 HASH functions . . . 27
13 Block ciphers . . . 30
14 DLog and Diffie-Hellman . . . 32
15 RSA . . . 36
16 Weak keys . . . 38
17 Randomness . . . 40
18 Cryptographic primitives . . . 43
19 Digital signature . . . 44
20 Secret-Sharing . . . 46
21 Sketch on more advanced signature schemes . . . 49
22 Introduction to E-Payment Systems . . . 50
23 EMV . . . 52
24 CNP and Traditional Internet Payment . . . 56
25 Online banking . . . 57
26 Mobile Banking . . . 58
27 PayPal . . . 59
28 Mobile Payment . . . 60
29 PCI . . . 62
30 E-commerce . . . 63
31 Post Quantum Cryptography . . . 65
32 Zero knowledge algorithm . . . 66
II APPLICATIONS . . . 67
33 Passwords & Authentication . . . 68
34 Public Key Infrastructure . . . 74
35 TLS . . . 85
36 SAML and co . . . 96
37 OAuth and OpenID Connect . . . 111

Part I
CRYPTOGRAPHY

1 Introduction

The goal of cryptography, as suggested by the name (*kryptós*= "hidden" + *graphia*="writing") is to enable a secure communication between two users, making it impossible for an eavesdropper to understand the information being exchanged. We call the legitimate users **Alice** and **Bob**, and we refer to the eavesdropper as **Eve**.
A channel is any physical or logical medium of communication from one user to another. A channel is called *secure* if the information exchanged over it cannot be overheard or tampered with by eavesdroppers. A channel which is not secure is called *insecure*. Usually in cryptography we assume that the channel is insecure. The intent of cryptography is to obtain a secure communication even using an insecure channel. Secure channels established by other means are sometimes employed in communication (as for example trusted couriers, personal contact between communicating parties, dedicated link of communication, ...) but they are too expensive to be frequently used in common scenarios.
To obtain a secure transmission Alice and Bob do not exchange their messages on the channels directly, but they transmit the messages in a disguised form. We call:

- **plaintext**: the original message that Alice and Bob want to exchange
- **ciphertext**: the disguised message transmitted over the channel.

We use $\mathcal{P}$ to indicate the set of plaintexts and $\mathcal{C}$ for the set of ciphertexts.

Let us consider two functions:

- Encryption function: f : $\mathcal{P} \to \mathcal{C}$
- Decryption function: g : $\mathcal{C} \to \mathcal{P}$

such that $g \circ f = 1\mathcal{P}$.

Thanks to f and g we can describe how a secure communication works:

- *Alice* wants to send the message $m \in P$ to *Bob*;
- *Alice* computes $f(m) = c \in \mathcal{C}$;
- *Alice* sends c to *Bob* through the channel;
- *Bob* computes $g(c) = g(f(m)) = (g \circ f)(m) = 1\mathcal{P}(m) = m$, finding the original message.

From the definitions of encryption and decryption functions, we have that f is injective and g is surjective.

Example: $\mathcal{C} = \{0, 1, 2\}, \mathcal{P} = \{YES, NO\}$

$$\begin{array}{ll} f : \mathcal{P} \to \mathcal{C} & g : \mathcal{C} \to \mathcal{P} \\ YES \to 0 & 0 \to YES \\ NO \to 1 & 1 \to NO \end{array}$$

Observe that 2 is in the ciphertext set, although it does not correspond to any plaintext. Alice and Bob might send 2 from time to time in order to puzzle Eve.

Alice and Bob choose together f and g. However, this choice is not without difficulties. In fact:

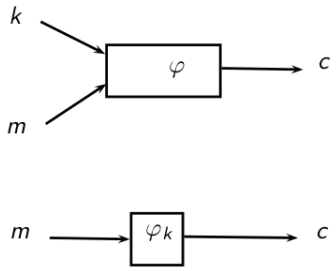
- $\mathcal{P}$ e $\mathcal{C}$ must not be **too small** in order to avoid an exhaustive search (brute force) attack, otherwise Eve can try all the plaintext-ciphertext pairs, (m, c);
- even if $\mathcal{P}$ and $\mathcal{C}$ are huge, f and g could have such an easy description that Eve could try to guess them; so f and g must be difficult to find.

The most important problem in this scenario is the following: f and g must be kept secret so that Eve cannot know them $\rightarrow$ f and g must be exchanged on a secure channel. All these difficulties make it **unfeasible** for Alice and Bob to **agree** directly and secretly on the encryption and decryption functions, since they would spend too much in terms of performance of the channel.
To solve these problems at least partially we introduce the idea of a **cryptographic key**. We use $\mathcal{K}$ to indicate the set of keys: $\mathcal{K} = \{\text{keys}\}$.

Let us consider the map $\varphi : \mathcal{P} \times \mathcal{K} \rightarrow \mathcal{C}$ such that for any key $k \in \mathcal{K}$, then $\omega(m, k)$ is an encryption function:

$$\varphi(m, k) = f(m), \qquad \text{with} f : \mathcal{P} \rightarrow \mathcal{C}$$

Usually, we also write $\varphi_k(m) = \varphi(m, k)$. We have built a set of encryption functions indexed by the keys in $\mathcal{K}$. Sometimes we call φ a **cipher**.



The fundamental differences are:

1. φ can be an extremely complicated algorithm but it is known to everyone, and the choice of it can safely be established through an insecure channel;

2. the single part of the system that must be kept secret (and thus exchanged over a secure channel) is the key k, which determines the encryption function to use.

From now on, we say that an information is **secret** or **private** if it must be communicated over a secure channel, otherwise it is public. Obviously, if we are able to keep φ secret as well we obtain greater secrecy, provided that we are able to evaluate the robustness of the cipher.
Kerckhoffs's principle:

> *A cryptosystem should be secure even if everything about the system, except the key, is public knowledge.*

The **advantages** of exchanging only the key k rather than the whole encryption function φ are that:

- it is easier to keep secret k than φ
- if the key is discovered, it is sufficient to change k and we do not need to renegotiate φ

The main **disadvantage** is that the attacker can break the system just by finding k.

2 Attacks

Up to now we focused on the legitimate users, Alice and Bob. Now we look at Eve. Depending on the possibilities of the attacker, we can distinguish three kinds of attacks:

- CIPHERTEXT-ONLY
- KNOWN-PLAINTEXT
- CHOSEN-PLAINTEXT

2.1 CIPHERTEXT-ONLY

Let

$$\{m_1, \dots, m_N\} \subset \mathcal{P}$$

be some plaintexts. Let us suppose we have fixed a key k in $\mathcal{K}$. So, we have an encryption function, φ_k , corresponding to k. We encrypt any plaintext with φ_k , obtaining the corresponding set of ciphertexts:

$$\{c_1, \dots, c_N\} \subset \mathcal{C}, \quad c_i = \varphi_k(m_i)$$

$$\{m_1, \dots, m_N\} \xrightarrow{\varphi_k} \{c_1, \dots, c_N\}$$

Let us suppose that Eve knows $\mathcal{P}$ and $\mathcal{C}$, but she obtained also some ciphertexts: $\{c_1, \dots, c_N\}$. Her aim is to decrypt at least one ciphertext. Note that sometimes Eve does not need to know the cipher φ. In this case, the information of Eve is too small to obtain any plaintext, but it could be enough to get some information on the plaintexts. For example, suppose that $c_1 = c_5$. Although Eve cannot get m_1 or m_5, she is able to conclude that the plaintexts corresponding to c_1 and c_5 are the same: $m_1 = m_5$. If Eve knows more, then she can get more. In particular, Eve can obtain more information if she knows the probability distribution of the plaintexts.

Example

$$\mathcal{P} = \{1, 2, \dots\}$$

We suppose that the plaintext 1 is transmitted very often, $P(m = 1) = 99\%$, while the other plaintexts are transmitted rarely. We consider as set of ciphertexts a set which contains the letters of the alphabet. Eve is intercepting the traffic and notes the following string of symbols:

xxxxaxxxAxxxxxxxxxxxxcxxx

What can Eve deduce? Eve guesses that x, being the most probable cipher, corresponds to the plaintext 1, since 1 happens more often than the others. Thus she concludes that

$$x = f(1)$$

We note that even if Eve does not know anything about the other plaintexts (like those corresponding to the ciphertexts a, A, c in the example string), she is still able to recover the 99% of the message. Summarizing, if an attacker knows the probabilty distribution of the plaintexts, she could obtain a lot of information merely by observing some ciphertexts. However, do not forget our setting: all plaintexts are encrypted with the same cipher and the same **key**.

2.2 KNOWN-PLAINTEXT

In a known-plaintext attack Eve has obtained, somehow, a set of ciphertexts $\{c_1, \dots, c_N\}$ and their corresponding plaintexts $(m_1, c_1), \dots, (m_N, c_N)$. It is also customary to assume that Eve has full knowledge of the cipher. With this extra information Eve might aim at something more interesting than some decryptions. Her main goal becomes to **recover the key**. This kind of attack is called a Key-Recovery attack. When Eve gets the key, she will be able to encrypt and decrypt any message, since she will have the same information shared by Alice and Bob. The possession of plaintext-ciphertext pairs can be insufficient to get the key in a reasonable time with

Eve's resources. However, Eve does not need to recover the key in order to break the system. An alternative goal for Eve is to discover a functionally equivalent algorithm for encryption and decryption. In other words, Eve might write down an algorithm whose output is the same as that of φ_k , even without getting k. This kind of attack is known as **Global Deduction** (or Global Reconstruction).

2.3 CHOSEN-PLAINTEXT

In a chosen-plaintext attack, the attacker can obtain the ciphertexts corresponding to an arbitrary set of plaintexts $\{m_1, \ldots, m_N\}$ of her own choosing. Also in this case the attacker can have two different goals: to recover the key or to mount a Global Deduction attack. In this scenario, the attacker can be facilitated in her attack from a suitable choice of the plaintext-ciphertext pairs. Sometimes, both in the known-plaintext and in the chosen-plaintext attack, Eve may have a less ambitious goal: she tries to guess some plaintext-ciphertext pairs, which were not previously known.

2.4 Feasibility

An important question for the attacker is the feasibility of an attack. In a ciphertext-only attack, the attacker only needs to be able to intercept the traffic from the sender to the receiver. In a known-plaintext or a chosen-plaintext attack, she needs a deeper access to the communication infrastructure used by Alice and Bob. A scenario for a **known-plaintext** is, for example, a spy who can observe the information exchange by having a peek at Alice. For a **chosen-plaintext** attack we can think of a spy who is able to access a ciphering machine and to input a set of chosen plaintexts, while collecting the corresponding ciphertexts. The required attacker's skills are minimal for a ciphertext-only attack, medium for a known-plaintext attack and high for a chosen-plaintext attack. This suggests that, in general, a ciphertext-only attack is more difficult than a known-plaintext attack, which is on its own more difficult than a chosen-plaintext attack.

Skill of the attacker: Ciphertext-only $<$ Known-plaintext $<$ Chosen-plaintext

Effort of the attack: Ciphertext-only $>$ Known-plaintext $>$ Chosen-plaintext

3 Shannon's theorem

We recall some notation:

- $\mathcal{K}$ is the set of keys
- $\mathcal{P}$ is the set of plaintexts
- $\mathcal{C}$ is the set of ciphertexts
- φ is a cipher such that $\forall k \in \mathcal{K}$

$$\varphi_k = \mathcal{P} \to \mathcal{C} \quad \text{is an encryption function}$$

  φ_k is invertible: there exists a unique φ_k^{-1} so that $\varphi_k^{-1}(\varphi_k(m)) = m$.

We introduce the notation:

- if $m \in \mathcal{P}$, we use $P(m)$ to indicate the a priori probability that m occurs;
- if $k \in \mathcal{K}$, $P(k)$ indicates the probability that k is the chosen key;
- if $c \in \mathcal{C}$, $P(c)$ is the probability that c is the transmitted ciphertext;
- $P(\mathcal{P})$ is the probability distribution of the plaintexts;
- $P(\mathcal{C})$ is the probability distribution of the ciphertexts;
- $P(\mathcal{K})$ is the probability distribution of the keys.

If there is a plaintext $\bar{m}$ such that $\mathrm{P}(\bar{m}) = 0$, then it means that it never occurs and we can remove it from the set of the plaintexts. So, we can always assume that $P(m) > 0$ for every $m \in \mathcal{P}$. In the same way, we assume that $P(k) > 0$ and $P(c) > 0$ for any $k \in \mathcal{K}$ and any $c \in \mathcal{C}$.
The two probability distributions $P(\mathcal{P})$ and $P(\mathcal{K})$ determine the probability distribution $P(\mathcal{C})$, because **only one** ciphertext can be obtained using the encryption algorithm φ with a given plaintext and key. Hence:

Let x and y be two events.

1. The **joint probability** $P(x \wedge y)$ denotes the probability that x occurs **and** also y occurs.
2. The **conditional probability** $P(x|y)$ denotes the probability that x occurs **given that** y occurred.
3. We say that x and y are **independent** if $P(x \wedge y) = P(x)P(y)$.

3.1 Clever boy

From now on we assume that the keys and the plaintexts are independent:

$$\forall m \in \mathcal{P}, \forall k \in \mathcal{K} : P(m \wedge k) = P(m) \cdot P(k).$$

This assumption is known as *Clever boy assumption*.
What we are saying is that when Alice and Bob choose the key to use, their choice is made independently from the messages that they intend to transmit. This assumption is realistic because the key k is commonly chosen long before we know which message will be sent (if we already know the messages... it is useless to encrypt them!).

3.2 The perfect cipher, according to Shannon

Suppose that Eve can just intercept the ciphertexts. In the classical cryptographic situation, we are interested in the conditional probability $P(m|c)$ which is the probability that the plaintext m is sent, given that the ciphertext c is received.
A cipher is called **perfect** if

$$\forall m \in \mathcal{P} \quad \text{and} \quad \forall c \in \mathcal{C} \Rightarrow P(m) = P(m|c)$$

In other words, in a perfect cipher Eve cannot obtain any new information on the sent plaintext looking at only one ciphertext. In particular, although Eve knows the probability distribution of the plaintexts, she cannot get any information from c. Hence a perfect cipher is **unbreakable** if Eve can use **only one** ciphertext.

3.3 Shannon's theorem

Let us fix an integer n. Assume that the keys have n bits, the plaintexts have n bits and also the ciphertexts have n bits. Assume also that:

- the plaintexts and the keys are independent (Clever boy assumption)
- any n-bit string may happen as a key or as a plaintext

then φ is a perfect cipher if and only if both the following conditions hold:

1. the keys are perfectly random;
2. for any plaintext m and any ciphertext c there is one and only one key k such that c is the encryption of m with the key k.

3.3.1 Shannon's theorem: comments

Given **any** probability distribution of the plaintexts and even if such distribution is known, what Eve observes is a perfectly random cipher. Hence Eve cannot recover any information on the sent message from the ciphertext she intercepted. Beware that this is true only if Eve intercepted only one ciphertext. If Eve intercepts more ciphertexts (encrypted with the same key) then the Shannon theorem **no longer guarantees perfect secrecy**.
This explains why a perfect cipher is not unbreakable in a practical sense. A perfect cipher is unbreakable provided that the attacker can use only one ciphertext to break it. Instead, we would like an **ideal** cipher able to resist to any practical attack, even using several ciphertexts. Hence, **a perfect cipher is not necessarily ideal**.

3.4 The Vernam cipher

It may appear that the conditions of the Shannon theorem are too restrictive to have practical realizations, but this is not true. One well-known example of perfect cipher is the **Vernam cipher**, also known as **One-Time Pad**.
Let us take the same set for the keys, the plaintexts and the ciphertexts:

$$\mathcal{P} = \mathcal{C} = \mathcal{K} = (\mathbb{F}_2)^n$$

m is a vector of n bits:

$$m = (010 \ldots 1)$$

Given a key k, the encryption is done summing the key and the message bitwise in $(\mathbb{F}_2)^n$:

$$c = \varphi_k(m) = k + m$$

in other words, c is the XOR of the binary strings m and k.
The Vernam cipher is perfect, **once the keys are randomly chosen**. There are mainly three disadvantages in using the Vernam cipher:

1. randomness → the keys must be chosen perfectly at random;
2. key length → the length of the key must be equal to the length of the message; moreover the key must be exchanged secretly: such an operation is very expensive, and indeed if we could guarantee the security of an exchanged key of the same length as the message, we may as well simply exchange the message itself;
3. one-time → each key must be used for one encryption only. In fact, if Eve knows a single plaintext-ciphertext pair (m_1, c_1), she can easily recover the key:

$$m_1 \xrightarrow{k} c_1 = m_1 + k$$

$$m_1 + c_1 = m_1 + k + m_1 = k$$

Hence, if the key is not changed after each encryption, she can use the recovered key k to decrypt all the other ciphertexts.

4 From Shannon to modern cryptography

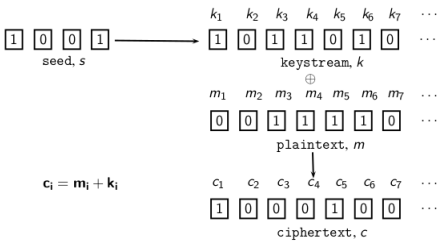
Modern cryptography tries to improve One-Time-Pad. Two different approaches were adopted, which led to two different types of modern ciphers. The two approaches are:

- To create very long keys (ideally of infinite length) to be summed with the plaintexts → **Stream Ciphers**
- To use the same key more than once in a way that does not compromise the cipher → **Block Ciphers**

4.1 Stream ciphers

Stream ciphers:

- take a small vector of a few (random) bits, called seed, which must be kept **secret**
- obtain a very long sequence of **pseudorandom** bits, called keystream
- sum the keystream with the plaintext bitwise, getting the ciphertext



For a good stream cipher, Eve, even possessing the keystream:

- should not be able to recover the **seed**
- should not be able to predict the **rest of the keystream**

4.2 Block ciphers

In a block cipher any plaintext m and any key k correspond to a ciphertext. Also, once the key k is chosen, different plaintexts correspond to different ciphertexts:

$$\begin{cases} \varphi_k(m_1) = c_1 \\ \varphi_k(m_2) = c_2 \\ \quad \vdots \\ \varphi_k(m_N) = c_N \end{cases} \tag{1}$$

For a good block cipher, Eve, even using multiple plaintext-ciphertext pairs, should not be able to recover the **key**.
As already seen, we cannot achieve these goals using the One-Time-Pad cipher, but it is possible if the encryption function φ_k is very complicated. To keep Eve from recovering the key from the intercepted plaintext-ciphertext pairs, a block cipher uses a very complicated relation to link the key k and the encryption algorithm φ. Such greater complexity means that block ciphers are in general slower than stream ciphers, but they are considered more secure.

4.3 Stream-Block comparison

Traditionally we have the following comparison:

STREAM	BLOCK
Quick	Slow
Hardware-oriented	Software-oriented
Less secure	More secure

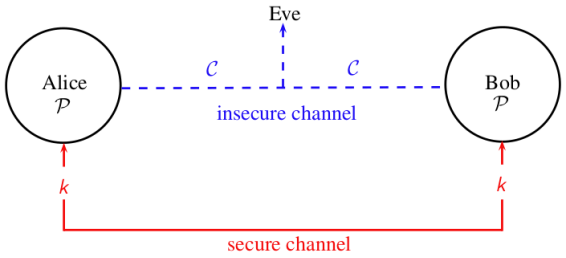
Some examples of the most common stream ciphers and block ciphers are:

Type	Cipher	Applications
Stream	A5/1, A5/2, A5/3 RC4 E0	Phone (GSM) Internet Bluetooth
Block	DES 3DES AES PRESENT	(old) Smart cards Everywhere sensor networks

Recently (2011, last version) a new stream cipher, ZUC, was developed by DACAS (China), whose design is very different from classical stream ciphers. ZUC is slower and may be more secure than the others.

4.4 Symmetric cryptography

All the cryptographic systems presented up to now (one-time-pad, block ciphers, stream ciphers) assume that Alice and Bob agreed on a secret key before any ciphertext is transmitted. Such keys must be exchanged on a secure channel. This kind of cryptography, where Alice and Bob



have access to a shared secret (the key) that allows secure communication, is called **symmetric cryptography** (or secret key cryptography).

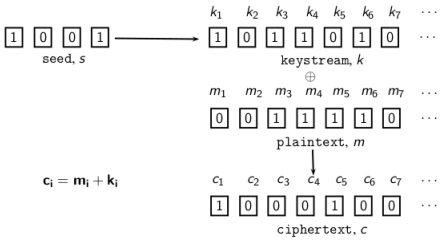
Crucial points and questions arising

- The key must be randomly generated as long as possible (is it possible?)
- Alice and Bob know the encryption key (how can we do that??)
- Only Alice and Bob (and, of course, who generates the key) know the encryption key
- Must we always suppose that Eve knows the probability distribution of the plaintexts ??
- Is the One time pad used nowadays at all?

5 Stream ciphers

We already introduced stream ciphers. We recall the idea on which they are based:

- They take a small vector of a few random bits, called seed, which must be kept secret
- They obtain a very long sequence of pseudorandom bits, called keystream (or output sequence)
- They sum the keystream with the plaintext bitwise, getting the ciphertext

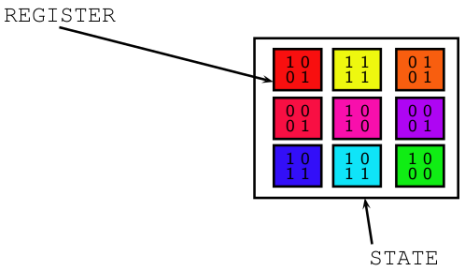


The main components of a stream cipher are:

- the states
- the key loading
- the update function
- the output function

The states

A state for a stream cipher is a vector of bits, so we suppose it belongs to $(\mathbb{F}_2)^N$, where N is a fixed positive integer ($N \in \mathbb{N}$). The bits composing the state are organized in substructures, called registers.

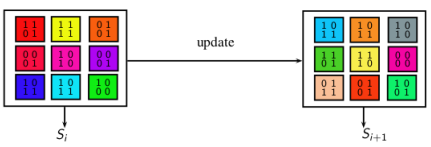


The update function

Starting from any state, it is possible to obtain a new state using the update function:

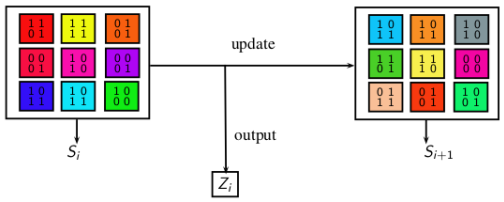
$$(\mathbb{F}_2)^N \xrightarrow{update} (\mathbb{F}_2)^N$$

We can see the update function as a clock function, which at the i-th step maps the state S_i to the state S_{i+1}.



The output function

From any state we extract typically one bit Z_i, using the output function.



Concatenating all the bits obtained from the output function, we get the keystream.

The key-loading

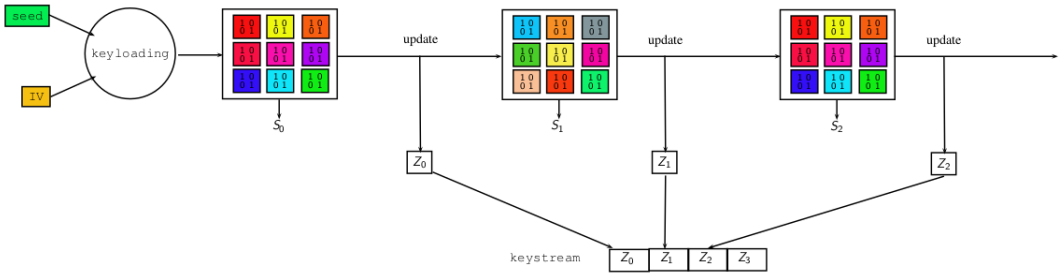
To apply the update function we need an initial state, which is obtained from the secret seed and a public vector, called initialization vector (IV), using the key-loading function.

A key loading function is a map:

$$kl : (\mathbb{F}_2)^m \times (\mathbb{F}_2)^n \longrightarrow (\mathbb{F}_2)^N$$

$$(IV, s) \to S_0$$

where S_0 is the initial state.



If Eve is able to recover the states, then she breaks the cipher. For this reason, a good stream cipher does not allow Eve to find the states by knowing part of the keystream.

5.1 Warm up

We note that the first bit in output, $Z_0, Z_1, Z_2, \ldots$ strongly depend on the initial state S_0 ; hence, knowing these it could be easy to recover S_0. To overcome this weakness, it is usual to precede the encryption with a **warm-up-phase**, which consists in applying update functions many times, without outputting any bits of keystream. Once the warm-up phase is concluded, we can start to output the bits of the keystream. The warm-up phase allows us to avoid a lot of practical attacks (e.g. RC4). It is a good practice that we recommend.

A note

Observe that if there is S_i such that update$(S_i) = S_i$, then

$$S_i = S_{i+1} = S_{i+2}$$

We want to avoid this situation, because in that case the keystream becomes constant and Eve can easily break the system and recover all the subsequent plaintext.

Another note

Given any initial state S_0, the states are periodic, since they are in a finite number and at some point we will obtain again one of the previous states. Consequently, the keystream is periodic too. We cannot avoid that.
The smallest number i (number of repetitions of the *update* producte), such that

$$(update \circ \cdots \circ update)(S_0) = S_0$$

is called period of the register. Note that, in general, the period depends on the initial state. We require the period of the register to be quite large, whatever the initial state is, and this can only be achieved by a well designed update function.

An example

We consider states composed of one register with three bits. All the possible states are the vectors of $(\mathbb{F}_2)^3$:

STATES
(000)
(100)
(010)
(001)
(110)
(011)
(101)
(111)

We have to define an update function:

$$f : (\mathbb{F}_2)^3 \to (\mathbb{F}_2)^3$$

Te update function that we take is:

$$(x, y, z) \mapsto (y + z, x, y)$$

In other words, to pass from a state to another we:

- shift to the right the bits in the first two positions (from left to right)
- in the first position we put the sum of the last two bits of the input

x	y	z	$\xrightarrow{f}$	y+z	x	y
1	0	1		1	1	0

We can describe the previous operations in a compact form, using the feedback polynomial:

$$x^3 + x + 1 = x^3 + 0 \cdot x^2 + 1 \cdot x + 1 \cdot 1$$

The degrees decrease from left to right. With this notation we indicate that, given the state (a_2, a_1, a_0), the next state is (a_3, a_2, a_1), where $a_3 = 0 \cdot a_2 + 1 \cdot a_1 + 1 \cdot a_0$. Proceeding in this manner we obtain the other states:

$$(101) \to (110) \to (111) \to (011) \to (001) \to (100) \to (010) \to (101)$$

Note that if the initial state is (000) we have:

$$(000) \to (000)$$

Thus (000) is not a good initial state and we discard it from all possible initial states to choose (fortunately, it is not possible to obtain the state composed of all zeros starting from a state

different from zero).
Suppose we have chosen $x^3 + 1$ as feedback polynomial instead of $x^3 + x + 1$. We have:

$$f : (\mathbb{F}_2)^3 \to (\mathbb{F}_2)^3$$

$$(x, y, z) \mapsto (z, x, y)$$

The function f maps the state (a_2, a_1, a_0) to the state (a_3, a_2, a_1), where $a_3 = 0 \cdot a_2 + 0 \cdot a_1 + a_0$. If we choose as initial state (101), we get:

$$(101) \to (110) \to (011) \to (101)$$

and so we have only three distinct states. Moreover, if the initial state is (000) or (111), we have:

$$(000) \to (000)$$

$$(111) \to (111)$$

not obtaining other states different from the initial one. Choosing another initial state, we observe a behaviour similar to (101).

$$(100) \to (010) \to (001) \to (100)$$

These considerations suggest that as feedback polynomial $x^3 + x + 1$ is better than $x^3 + 1$.

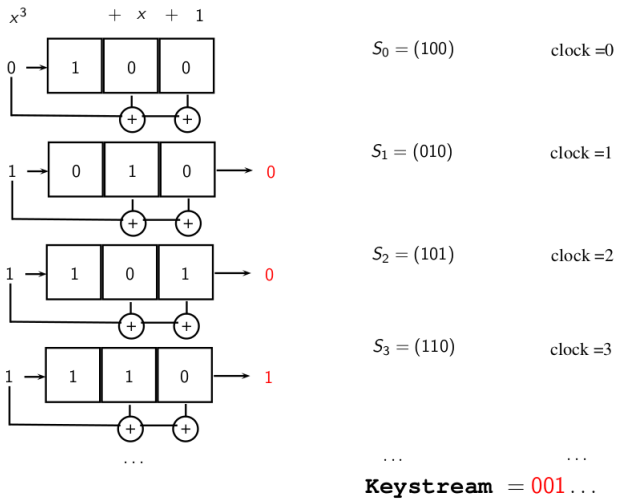
6 Linear Feedback Shift Registers

The previous examples show how a linear feedback shift register (LFSR) works. A LFSR of length n is a shift register composed by n bits. At any clock the following operations are executed:

- the last bit on the right is output and forms part of the keystream
- the other bits in the register are shifted to the right by one position
- the XOR of some bits of the register are put in the first position on the left.

The bits in the LFSR that influence the new input bit are called taps. The feedback polynomial of an LFSR of length n is a polynomial in $\mathbb{F}_2[x]$ of degree n whose non-zero terms of degree less than n correspond to the taps.

$n = 3$; $S_0 = (100)$; feedback polynomial: $x^3 + x + 1$



As already seen, if the initial state is the zero vector, we always obtain the zero state using an LFSR. The non-zero initial states are $2^n - 1$. When the initial state is different from zero, we want to obtain a number of distinct states which is as large as possible.

Theorem: For any register length n there is always at least one LFSR able to produce ALL non-zero states starting from any non-zero initial state.

The LFSR in the theorem is called "maximum-length LFSR" and its feedback polynomial is called "**primitive**".

6.1 LFSR: pseudorandom generator (PRNG)

Any binary sequence contains runs of 0 and 1. For example, the sequence 000110111001 is composed by 6 runs of length 3, 2, 1, 3, 2, 1, respectively. m-sequences have been studied extensively in cryptography because they **simulate** the behaviour of a **random** sequence very well.

Fact

Given a maximum-length LFSR of n bits and reading $2^n - 1$ consecutive bits of the m-sequence that it produces, we have that:

1. one half of the bits are 1 and one half are 0 (actually the 1's are one more than the 0's)
2. there are $2^n - 1$ runs:
   - 1/2 of the runs has length 1
   - 1/4 of the runs has length 2
   - ...
   - $1/2^i$ of the runs has length i (for $2 \leq i \leq n-2$)
   - there is only one run of $n-1$ zeros and none of the runs has $n-1$ ones
   - there is only one run of n ones and none of the runs has n zeros.

Example

We use as feedback polynomial the primitive polynomial x^5+x^2+1, hence the period is $2^5-1 = 31$ bits and we expect $2^5 - 1 = 16$ runs. Starting from the state (10000), the obtained m-sequence is

$$\underline{0000}\ \underline{1}\ \underline{00}\ \underline{1}\ \underline{0}\ \underline{11}\ \underline{00}\ \underline{11111}\ \underline{000}\ \underline{11}\ \underline{0}\ \underline{111}\ \underline{0}\ \underline{1}\ \underline{0}\ \underline{1}$$

There are indeed 16 runs, and

- 8 runs have length 1
- 4 runs have length 2
- 2 runs have length 3
- There is only 1 run of 4 zeros and none of 4 ones
- There is only 1 run of 5 ones and none of 5 zeros

6.2 LFSR: cryptographic weakness

Once the attacker obtains the feedback polynomial, she is able to reproduce the whole keystream from any (short) subsequence of the keystream having length n. An adversary may obtain the required subsequence of the keystream mounting a known-plaintext or chosen-plaintext attack. Hence LFSRs are vulnerable with respect to known-plaintext attacks and they must not be used as keystream generators (although they can be used as components).

Exercises

1. Let L be a (binary) LFSR of length 5 and feedback polynomial $x^5 + x + 1$ with initial state $S_0 = (0, 1, 0, 1, 1)$. Compute the first 5 bits of the keystream generated by L.
2. Let L be a (binary) LFSR of length 3 and feedback polynomial $x^3 + x^2 + x + 1$. Determine if L is a maximum-length LFSR.

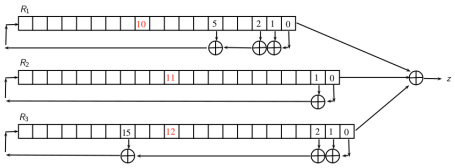
7 Cryptography in GSM: the A5 family

The GSM is de facto the standard protocol for 2G digital cellular networks used by mobile phones. Data are transmitted in blocks of 228 bits. It was the first standard to introduce cryptographic algorithms for security.
The standard includes 4 kinds of encryption:

- A5/0: no encryption
- A5/1: stream cipher based on LFSRs
- A5/2: stream cipher based on LFSRs
- A5/3: stream cipher based on the block cipher KASUMI

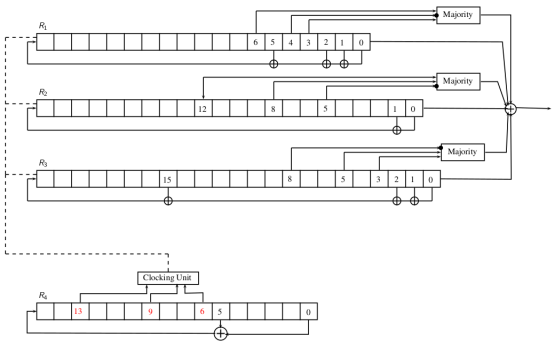
7.1 A5/1

- year: 1987
- keystream length: 228 bits
- number of LFSRs: 3
- update of registers: majority function on the control bits
- Registers: length of 19, 22, 23 (total: 64 bits of state)



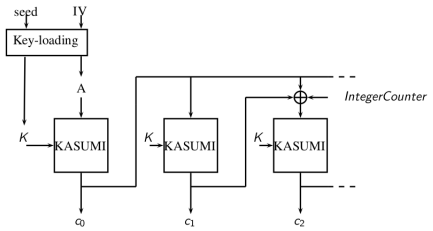
7.2 A5/2

- year: 1989
- keystream length: 228 bits
- number of LFSRs: 4
- update of registers: majority function on the control bits R_4
- Registers: length of 19, 22, 23, 17 (total: 81 bits of state)



7.3 A5/3

A5/3 is the last stream cipher of the A5 family and provides users with a higher level of security than A5/1-2. It is used in GSM, UMTS (f8-UEA1, f9-UIA1) and GPRS (GEA3) communications systems. It is based on the block cipher KASUMI.



KASUMI is a block cipher with

- space of the keys : $\mathcal{K} = (\mathbb{F}_2)^{128}$;
- space of the messages : $\mathcal{P} = (\mathbb{F}_2)^{64}$.

Hence for any $k \in \mathcal{K}$ there is an encryption function

$$\phi_k : (\mathbb{F}_2)^{64} \to (\mathbb{F}_2)^{64}$$

The output function extracts some of the bits of the KASUMI ciphertext c_i, with $i \neq 0$. For example, since the keystream consists of 114 bits and each ciphertext is composed of 64 bits, a typical choice is:

- to take the first 64 bits of a ciphertext
- to concatenate them with 50 bits of the next ciphertext

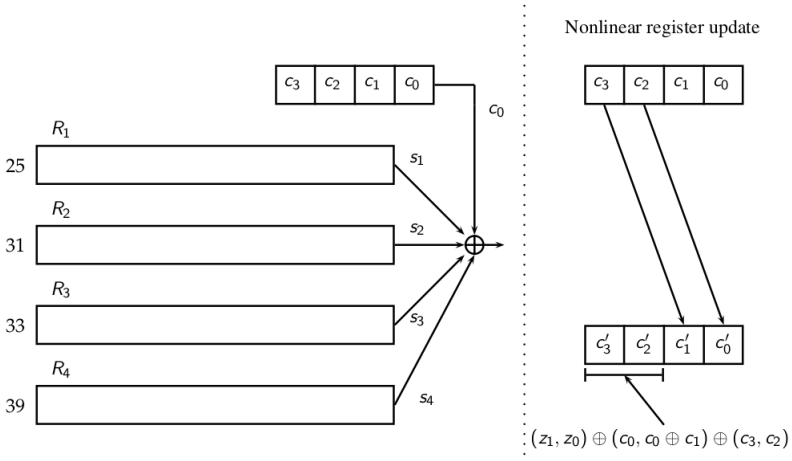
7.4 A5/1-2-3: comments

- A5/2 is extremely weak and it can be broken in real time with inexpensive equipment; it is therefore no longer supported by new mobile phones.
- A5/1 is affected by a number of serious weaknesses, and its use is strongly discouraged, since there are practical attacks that can break the cipher.
- A5/3 is the common standard for the new generation of mobile and it is considered secure, even though there do exist practical attacks to KASUMI that suggest some significant weaknesses of the cipher.

8 Cryptography in Bluetooth: E0 and RC4

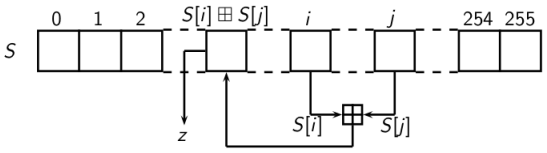
8.1 E0

- year: 1999
- used in Bluetooth standard
- number of LFSRs: 4
- Linear registers: length of 25, 31, 33, 39 (total for the linear part: 128 bits)



This is a good technique to construct stream ciphers, since it allows to reach high non-linearity using easy and fast computations.

8.2 RC4



We recall that during the key-loading we compute

$$j = j \boxplus S[i] \boxplus k[i']$$

This computation reveals that there is a strong correlation between the first byte of the keystream and some bytes of the key. An attacker can use this property to attack RC4, trying to recover the key or part of it. To avoid such a weakness, a warm-up phase usually precedes the production of the keystream. RC4-drop768 and RC4-drop1032, which are variants of classical RC4, use a warm-up phase that discards the first 768 and 1032 bytes of output, respectively.

RC4 does not specify how to construct the session key from the seed and the initialization vector. A classical choice is to use a session key of the form k = (IV || seed). Unfortunately, this is a wrong choice, since there are attacks based on this structure for the session key.

RC4 is widely used in popular protocols, especially in wireless protocols, WEP and WPA (but also TLS). There are dangerous practical attacks to WEP. Nowadays WPA2 is used for wi-fi, which is no longer based on stream ciphers but on block ciphers. To our knowledge, no practical attacks are known if RC4 is used with a careful choice of the key and a suitable warm-up phase.

9 DES-3DES

Data Encryption Standard (DES) is a block cipher based on Feistel encipherment, developed by IBM.

9.1 Feistel cipher

A Feistel cipher is a multi-round cipher that divides the current internal state of the cipher into two parts and operates only on a single part in each round of encryption or decryption. Feistel encipherment is used by several block ciphers. The input is a plaintext of 2w bits and a secret key K , which is divided in subkeys in the key schedule. The plaintext is divided in two blocks L_0, W_0 of w bits length. n rounds are performed and at the end the ciphertext is a 2w bits ciphertext. For decryption, the rounds are performed in inverse order.

Generic round i

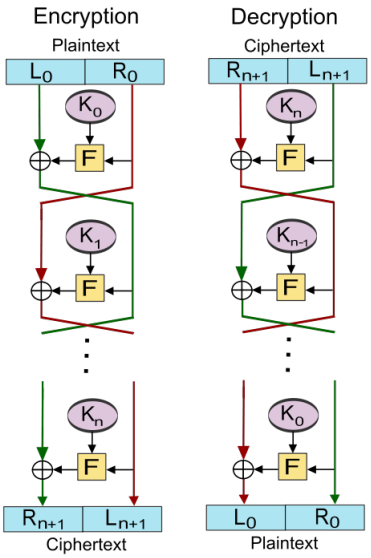
INPUT: L_{i-1}, R_{i-1} and the subkey K_i.
OUTPUT: $L_i := R_{i-1}$ and $R_i := L_{i-1} + F(R_{i-1}, K_i)$, where F depends on the chipher using Feistel encipherment.

Last round

The final output is the couple (R_n, L_n), resulting from the last round.

Decryption

It is very similar to encipherment: the input is the ciphertext and the application order of the subkeys is reversed, from K_n to K_1 .



For DES, there are $n = 16$ rounds and the blocks are 64 bits long. The key is 64 bits long: 56 bits are randomly generated and are employed in the algorithm. The last 8 bits are used in the error detection. The plaintext is permuted with a permutation IP, elaborated in the Feistel structure and then IP^{-1} is performed on it. The $n = 16$ rounds depend on a function F , operating in two blocks, one of 32 bits and one of 48 bits, to produce a 32-bits block. The function is composed by:

- an expansion function E which transforms a 32 bits block to a 48 bits block;
- the resulting block E (R) is XORed with the round key;
- $E(R) \oplus K$ is divided in 8 blocks of 6 bits: $B_1, \ldots, B_8$;
- $B_1, \ldots, B_8$ are processed by 8 S-boxes;
- a permutation is applied.

In order to decrypt the scheme is the same, but the keys are in inverse order.

3DES has been developed to increase the security of DES, since the key space of DES is too small for modern computers, which can explore it fully in a few hours. It consists in applying DES three times. The key, called key bundle consists of three keys k1, k2, k3 of 56 bits each.

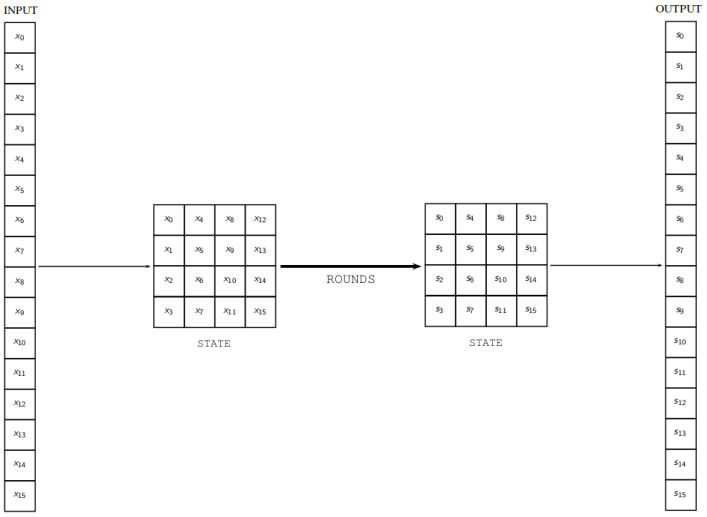
10 Advanced Encryption Standard (AES)

The plaintext of AES is of length 128 bits, $\mathcal{P} = (\mathbb{F}_2)^{128}$. There are three variants of AES:

VERSION	KEY LENGTH	N. ROUNDS
AES-128	128	10
AES-192	192	12
AES-256	256	14

For simplicity, we consider only the variant AES-128 with 128 bits. In other words, we have $\mathcal{P} = \mathcal{K} = (\mathbb{F}_2)^{128}$. We can see an element of $(\mathbb{F}_2)^{128}$ as an element of $(\mathbb{F}_{256})^{16}$, which means it is composed by 16 bytes ($128/8 = 16$).
A state of AES:

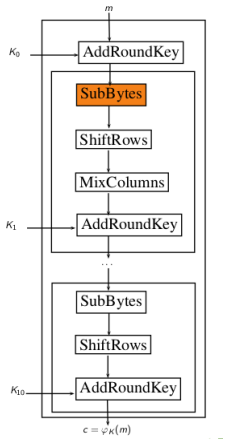
- is the partial cipher resulting from the application of a round function.
- can be depicted as a 4×4 matrix composed by 16 bytes.



AES consists in an inital step of **AddRoundKey**, followed by different rounds of these 4 steps:

- **SubBytes**;
- **ShiftRows**;
- **MixColumns** (not in the last round);
- **AddRoundKey**.

10.1 AES: SubBytes



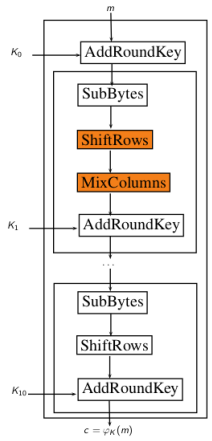
The SubBytes transformation provides **non-linearity** and is the application of the same map γ (called S-Box) to each byte:

$$\gamma : \mathbb{F}_{256} \to \mathbb{F}_{256}$$

$$s_i \mapsto s_i'$$

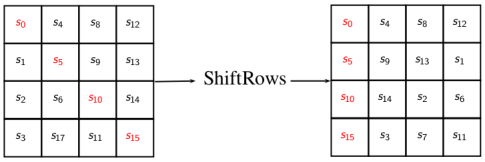
for $0 \leq i \leq 15$. In AES, SubBytes is the name of the S-Box, a byte-based look-up table with 256 entries. For example, the byte (00000000) is sent to the byte (11000110). Don't worry on how it's been computed.

10.2 AES: ShiftRows



In ShiftRows, bytes in the last three rows of the State are cyclically shifted (towards the left) with increased shift:

- the second row is shifted by one;
- the third row is shifted by two;
- the fourth row is shifted by three.

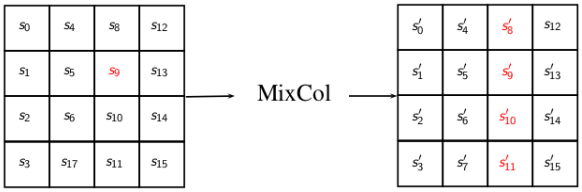


10.3 AES: MixColumns

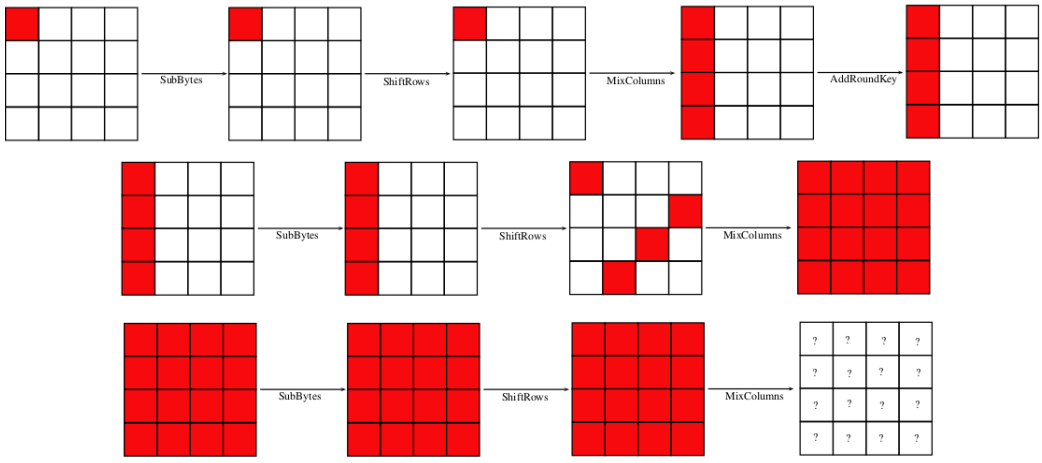
Each column of the state is multiplied (in the AES field) by the same matrix, which is the following:

$$\begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} = \begin{pmatrix} e & e+1 & 1 & 1 \\ 1 & e & e+1 & 1 \\ 1 & 1 & e & e+1 \\ e+1 & 1 & 1 & e \end{pmatrix}$$

For example for $\begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} s_8 \\ s_9 \\ s_{10} \\ s_{11} \end{pmatrix}$:

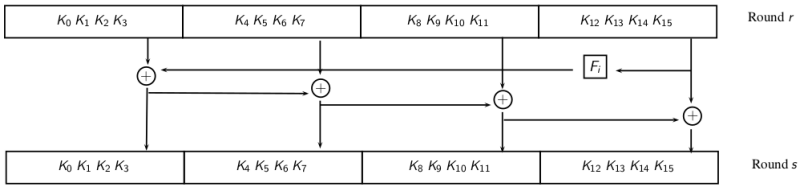


Three rounds



10.4 AES: AddRoundKey

Each byte of the state is combined with a block of the round key using bitwise XOR.



We show only the key schedule of the AES-128 version (the others are similar but more complex). The round key of round 1 is computed from the round key of round 0, that of round 2 from that of round 1 and so on, all by the same rule, which is described in the figure. Do not worry about the function F_i: it is a special method to mix four bytes together.

10.5 AES: MixingLayer

The map $\lambda : (\mathbb{F}_{256})^{16} \to (\mathbb{F}_{256})^{16}$ is a composition of two linear operations: ShiftRows and MixColumns. These two steps provide full diffusion. We can see the whole Mixing Layer as a matrix $\mathbf{M}$: the order of $\mathbf{M}$ is equal to 8 (i.e. $\mathbf{M}^8 = Id$). The low order of the Mixing Layer can be used to reveal some weakness of the cipher. On the contrary, the order of the Mixing Layer of SERPENT is greater than 2^{116}.

10.6 AES: best attacks

Key	Rounds	Texts	Time	Type	Year
128	5	2^{11}	2^{40}	Square attack	1998
128	5	$2^{29.5}$	2^{31}	Impossible diff.	2000
128	5	2^{39}	2^{39}	Boomerang attack	2004
128	6	2^{32}	2^{72}	Square attack	1998
128	6	$2^{34.6}$	2^{44}	**Partial Sum**	2000
128	6	$2^{91.5}$	2^{122}	Impossible diff.	2001
128	6	2^{71}	2^{71}	Boomerang attack	2004
128	7	$2^{128} - 2^{119}$	2^{120}	Partial Sum	2000
128	7	2^{32}	2^{128}	Collision	2000
128	10	2^{88}	$2^{126.1}$	BKR	2011
192	7	$2^{91.2}$	$2^{139.2}$	Impossible diff.	2008
192	8	2^{127}	2^{188}	Partial Sum	2000
192	10	2^{124}	2^{183}	Rectangle	2005
192	12	2^{123}	2^{176}	**Ampl. Boomerang**	2009
192	12	2^{80}	$2^{189.7}$	BKR	2011
256	8	2^{32}	2^{194}	Partial Sum	2000
256	9	2^{85}	2^{126}	Partial Sum	2000
256	10	2^{48}	2^{45}	Related Key	2010
256	10	2^{114}	2^{173}	Rectangle	2005
256	14	2^{40}	$2^{254.4}$	BKR	2011

The first key-recovery attack to AES-128, AES-192, AES-256 was published recently (2011). In this attack, proposed by Andrey Bogdanov, Dmitry Khovratovich and Christian Rechberger, a novel technique of block cipher cryptanalysis with bicliques has been exploited, which is 4 times more efficient than brute force. In particular, the BKR attack recovers the key of

- AES-128 in $2^{126.1}$ (k = 128, k = 126) operations;
- AES-192 in $2^{189.7}$ (k = 192, k = 190) operations;
- AES-256 in $2^{254.4}$ (k = 256, k = 254) operations.

Due to its extremely high cost, the proposed attack is still impractical but it proves that AES is NOT ideal. It is still the BEST-KNOWN attack on the full AES.

11 Serpent & Present

We recall the parameters for block ciphers AES, SERPENT and PRESENT. We take $\mathcal{P} = (\mathbb{F}_2)^n$, $K = (\mathbb{F}_2)^k$ and N is the number of rounds. The usual parameters are:

Block cipher	n	k	N
AES	128	128, 192, 256	10, 12, 14
SERPENT	128	128	32
PRESENT	64	80	31

Note that $k \geq n$.

11.1 SERPENT

Let $\mathcal{P} = \mathcal{C} = (\mathbb{F}_2)^n$, with $n = 128$. We consider $\mathcal{K} = (\mathbb{F}_2)^l$, with fixed length $l = 128$ (although the key is designed with variable length). The encryption consists of $N = 32$ similar rounds and works as follows:

- As usual, before the real rounds start we perform a XOR with a key (**whitening key**);
- $N - 1$ rounds with the same structure are applied, but using a different permutation, each composed of a XORing with the round key σ_k , a parallel S-Box γ and a linear mixing layer λ (we denote the round ρ by Round ρ, with $\rho = 1, \ldots, 31$);
- the last round (Round 32) is atypical and consists only of the S-Box and the mixing layer.

In Serpent, there are eight different S-Boxes, that change at every round, cycling every 8 rounds. So at round 1 we use the first S-Box, at round 2 the second S-Box, etc, but when we arrive at round 9 we use the first S-Box again and so on. All eight S-Boxes are nibble-base look-up tables, each with 16 entries.
For each input bit the corresponding output bit is as follows:

Input	S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7
0	3	15	8	0	1	15	7	1
1	8	12	6	15	15	5	2	13
2	15	2	7	11	8	2	12	15
3	1	7	9	8	3	11	5	0
4	10	9	3	12	12	4	8	14
5	6	0	12	9	0	10	4	8
6	5	5	10	6	11	9	6	2
7	11	10	15	3	6	12	11	11
8	14	1	13	13	2	0	14	7
9	13	11	1	1	5	3	9	4
10	4	14	14	2	4	14	1	12
11	2	8	4	4	10	8	15	10
12	7	6	0	10	9	13	13	9
13	0	13	11	7	14	6	3	3
14	9	3	5	5	7	7	10	5
15	12	4	2	14	13	1	0	6

For example, S_0 is sending (0000) to (0011) and S_7 is sending (0011) to (0000).

11.2 PRESENT

Lightweight Cryptography

The term lightweight cryptography refers to cryptographic algorithms that:

- are tailored to constrained devices
- are not intended for use against extremely strong adversaries

This does not mean weak cryptography. Lightweight cryptography was born essentially because:

- small computing devices are becoming more and more popular and represent a growing part of the pervasive communication infrastructure;
- the standard algorithms are too expensive for very small devices and protect data more strongly than necessary for typical use.

Important design principles of lightweight ciphers are:

- efficient hardware implementation;
- good performance;
- moderate security level.

In order to speed up the algorithm we want as few rounds as possible. In designing a lightweight block cipher we have to take into account some technical factors:

- to implement finite fields in hardware is expensive;
- Block size of 128 is too much;
- We do not have to keep information secret forever.

For these reasons, most modern lightweight block ciphers have:

- relatively small block size: 32, 48 or 64 bits;
- relatively small key size: 80 bits are often enough.

Both SP Network and Feistel can be used as designs for a lightweight block cipher.

An incomplete list of modern lightweight block ciphers

- 2005: mCrypton, STEA, ...
- 2006: Hight, SEA ...
- 2007: Clefia, Kasumi, DESL, DEXL, PRESENT, ...
- 2008: Puffin, ...
- 2009: KATAN, KTANTAN, MIBS, ...
- 2010: PRINT, ...
- 2011: Klein, LED, Twine, EPCBC, ...

PRESENT: overview

PRESENT is an iterated block cipher (translation based):

- number of rounds: 31
- plaintext space: $\mathcal{P} = (\mathbb{F}_2)^{64}$
- key space: $\mathcal{K} = (\mathbb{F}_2)^l$ with $l = 80$ or 128
- Round 0 consists of a parallel map $\gamma = 1_{\mathcal{P}}$, a linear transformation $\lambda = 1_{\mathcal{P}}$ and the translation $\sigma_{k^{(0)}}, where k^{(0)} \in (\mathbb{F}_2)^r$ is the first round key.
- A typical round consists of $\gamma\lambda\sigma_{k^{(\rho)}}$ where the non-linear operation γ is called sBoxLayer, the linear transformation pLayer and the addition with the round key is $\sigma_{k^{(\rho)}}$.

sBoxLayer

PRESENT uses only one 4-bit S-Box, as in the table below.

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$\gamma(x)$	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

Using a classification of all 4-bit S-Boxes due to Leander and Poschmann, the designers chose an optimal S-Box and in particular one well suited to have an efficient hardware implementation. Nevertheless, the S-Box of PRESENT is not optimal with respect to the refined classification (+ link).

pLayer

The pLayer has to maximize the diffusion and have a cheap implementation. In general, two possible solutions are possible:

- to use MDS matrices;
- to use binary permutation matrices.

PRESENT's pLayer employs the latter approach, hence it has less diffusion per round and it uses more rounds.
The affine map $\lambda : (\mathbb{F}_2)^{64} \to (\mathbb{F}_2)^{64}$ is a bit permutation defined by the following table, where the i-th bit of the vector is moved to the bit position $P(i)$ of the output.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P(i)$	0	16	32	48	1	17	33	49	2	18	34	50	3	19	35	51
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$P(i)$	4	20	36	52	5	21	37	53	6	22	38	54	7	23	39	55
i	32	33	34	35	36	37	38	39	40	41	41	43	44	45	46	47
$P(i)$	8	24	40	56	9	25	41	57	10	26	42	58	11	27	43	59
i	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
$P(i)$	12	28	44	60	13	29	45	61	14	30	46	62	15	31	47	63

The order of λ is equal to 3 (number of steps to return at the initial value).

Rounds	Key size	Attack	Complexity
5	80 bit	Algebraic	64 plaintext (Known Plaintext)
7	128 bit	Integral	2^{24} plaintext (Chosen Plaintext)
16	any	Differential	2 64 plaintext (Chosen Plaintext)
17	128 bit	RK Rectangle	2 63 plaintext (Chosen Plaintext)
24	any	SSA	2 64 plaintext (Chosen Plaintext)
26	128 bit	LH	2 64 plaintext (Known Plaintext)

The previous table shows that the number of rounds of PRESENT should not be dramatically reduced, in order to conserve a reasonable level of security.

12 HASH functions

A hash function, $h : \mathcal{X} \to \mathcal{Y}$ is a computationally efficient function mapping a string of arbitrary length (the message) to string of **fixed** length, called hash value, message fingerprint or message digest. Usually $X = (\mathbb{F}_2)^m$ and $Y = (\mathbb{F}_2)^n$, with $m >> n$.
Common choices in cryptography are $m \geq 2n$ and $n \geq 160$.

Applications

Hash functions are widely used in many applications, like:

- to verify the integrity of data
- in digital signature
- password verification
- key derivation
- pseudorandom number generators (PRNG)
- MACs

Classical definition

A hash function is a function h having at least the following properties:

- it is a compression function: h maps input x of arbitrary length in output $y = h(x)$ of fixed length n;
- it is easy to compute: given h and an input x, $h(x)$ is computed with a low computational cost.

Goal

Hash functions in cryptography are mainly used to guarantee data integrity. We want that:

- given inputs x, x', $x \neq x'$ (with x similar to x')
- given a hash function h
- $h(x) \neq h(x')$

Example:

```
input: "Roma"
output: DE5429D6F4FA2C86427A50757791DE88A0B75C85

input: "roma"
output: A6B6EA31C49A8E944EFE9ECBC072A26903A1461A
```

Properties

To consider a cryptographic hash function secure, it has to satisfy the following properties.

- **Preimage resistance**: given $h : \mathcal{X} \to \mathcal{Y}$ and $\bar{y} \in \mathcal{Y}$, it is computationally infeasible to find $\bar{x} \in \mathcal{X}$ such that $h(\bar{x}) = \bar{y}$.
- **Weak Collision (2nd-preimage) Resistance**: given $h : \mathcal{X} \to \mathcal{Y}$ and $\bar{x} \in \mathcal{X}$, it is computationally infeasible to find $\hat{x} \in \mathcal{X}$, $\bar{x} \neq \hat{x}$ such that $h(\bar{x}) = h(\hat{x})$.
- **Strong Collision resistance**: given $h : \mathcal{X} \to \mathcal{Y}$, it is computationally infeasible to find $\hat{x} \in \mathcal{X}$ e $\bar{x} \in \mathcal{X}$, $\bar{x} \neq \hat{x}$, such that $h(\bar{x}) = h(\hat{x})$.

From now on, when writing hash function we implicitly assume it to be a cryptographic hash function.
A good cryptographic hash function must satisfy the following requirements:

- one-way: it cannot be easily inverted, finding a preimage
- any collisions must be hard to find
- a second preimage must be hard to find
- the output must appear uniformly distributed

Hence, an ideal cryptographic hash function behaves as a **random function**, attaching to each possible input x a **random value** $h(x)$. If h is well designed, the only efficient way to determine the value $h(x)$ is to evaluate h in the input x. This should remain true even if many other digests $h(x_1), h(x_2), \ldots$ have already been computed.

Ideal hash functions

We can define (following the definition given by Bellare and Rogaway) a mathematical model of an ideal hash function.

Random Oracle Model

Suppose that $h : \mathcal{X} \to \mathcal{Y}$ is randomly chosen from all possible hash functions from $\mathcal{X}$ to $\mathcal{Y}$. The oracle is the only one to have access to h.

In other words, the oracle is like a **black-box** and no-one is able to compute the output of h using any algorithm or formula. The only way to compute the output of h is to query the oracle. Although a true random oracle does not exists in real life, we hope that a well designed hash function will behave like a random oracle.
Most hash functions are designed as iterative processes of a compression function f which has blocks of fixed size as inputs.

12.1 More on hash

In the iterated hash functions we have essentially three types of the compression functions:

- based on block ciphers (DES, 3DES, AES, ...)
- based on particular permutations (SPONGE, ...)
- based on arithmetic primitives (modular sums, etc.)

Examples of hash functions are:

- SNEFRU: collision attacks
- MD2: collision and preimage attacks
- MD4: collision, preimage and 2nd preimage attacks
- MD5: collision and preimage attacks
- SHA1: collision attacks
- SHA2 collision attacks on reduced versions
- SHA3 (KECCAC): new standard

12.1.1 Keyed hash functions

An important family of hash functions used in cryptography are those taking as input two parameters: the message and a secret key. They are called keyed hash functions. They are also known as **message authentication code** (**MAC**) because they provide data integrity and also authentication of data origin.
Suppose that Alice and Bob shared a secret key k, which identifies a hash function, denoted by h_k. Given the message x, both Alice and Bob can compute the corresponding digest $y = h_k(x)$. Alice transmits the pair (x, y) to Bob over an insecure channel. Bob receives the pair (x, y), computes $h_k(x)$ and verifies if $h_k(x) = y$. If $h_k(x) = y$, then Bob is confident that:

a) the message x was not altered

b) the message y was not altered

c) the sender of the message is Alice, since she is the only one, besides Bob, to know the secret key, k.

A message authentication code (MAC) algorithm is a family of functions h_k parameterized by a secret key k, with the following properties:

- it is a compression function: h_k maps an input x of arbitrary length to an output $y = h_k(x)$ of fixed length n
- it is easy to compute: for a known function h_k, given a value k and an input x, to compute $h_k(x)$ has a low computational cost.
- it is computation-resistant: given some pairs $(x_i, h_k(x_i))$, it is computationally infeasible to compute any pair $(x, h_k(x))$ for any new input $x \neq x_i$.

A common method to construct a MAC is to incorporate a secret key into a hash function, by including it as part of the message to be hashed. Such operation must be done carefully, since there are some attacks which can be carried out if an incorrect way is adopted to combine the message and the key.

Suppose an attacker knows a message x and its MAC $h_K(x)$. Then she can construct a valid MAC for another message (a forgery), without knowing K. This is clearly a security problem, because an attacker can be authenticated as a legitimate user in this way.

13 Block ciphers

13.1 Introduction

The composition of two simple functions can lead to a very complex permutation

> "Good mixing transformations are often formed by repeated products of two simple non-commuting operations. (C. Shannon, 1949)"

When we design a block cipher, we must choose components that do not commute; for example both AES and SERPENT would be trivially breakable if S-Box, MixingLayer and AddRoundKey would commute. If all the components are linear then the whole cipher is linear and hence easy to reconstruct, even without knowing the key. The same holds for affine components. The S-Box is the only non-linear part and its choice is crucial: using a weak S-Box leads to a great number of attacks.

- provided that the S-Box is well chosen, the role of MixingLayer becomes very important, because it is the only part that provides diffusion
- provided that the S-Box and MixingLayer are well chosen, the security of a block cipher lies in the secrecy of the key and the role played by the key in the round function

The number of rounds is approximately double the number of rounds that guarantees the absence of any measurable statistical difference from random output. There are some specialized statistical tests that try to check the random behaviour of a sequence generated for cryptographic purposes. The most famous test suite is the NIST suite. To test the random behaviour of a block cipher, people check the random behaviour of sequences generated using the cipher, e.g. concatenating the encryption of related plaintexts.
For an ideal block cipher, the best possible attack is the brute force attack. AES is not ideal, due to the biclique attack. Using a large number of rounds is "in general" considered to be good practice. Let us consider typical AES rounds; if it is possible to exchange an arbitrary number of round keys in a secure way, then it is possible to construct an ideal cipher, simply composing each time an arbitrary number of rounds. All translation-based block ciphers are susceptible to attacks similar to the Square attack (it can break easily any tb cipher up to the third round and it breaks AES up to the fourth round), especially if the MixingLayer does not provide a good diffusion. If you break a reduced version of a cipher, it does not imply that the full cipher is weak, but it suggests that attacks for the extended version may exist. The use of bad components leads to a weak cipher. On the other hand, the use of optimal components is no guarantee of obtaining a good cipher, since it is very difficult to predict the interplay of different components.

13.2 Questions

IS COMPOSING MORE THAN ONE BLOCK CIPHER A GOOD IDEA?

Composing more than one block cipher could be a good idea, depending on the scenario. If you want to do it, then:

- compose always at least three ciphers: with only two ciphers you are vulnerable to MITM (Man in the Middle, an attack that can be used on two combined block ciphers and which costs approximately the same as attacking one of them. You don't need to know it)
- do not compose many ciphers if you are not able to generate or to protect multiple keys
- do it if you are not confident on the robustness of one of the ciphers
- keep in mind that, in general, when composing more than one cipher we expect to have a more complex permutation, but that it is also possible to obtain a worse permutation. This subject is not well-defined: you are on your own

IS IT BETTER TO CHANGE THE KEY FREQUENTLY OR TO ALWAYS USE ONE SECRET AND ROBUST KEY?

Certainly do use the same key whenever:

- you do not have a good source of entropy
- you do not have frequent access to a secure channel
- you believe that your cipher is ideal

Frequently change the key if

- you think that Eve can break your cipher once she has intercepted enough messages
- you need to protect your data for a very long time

IS IT A GOOD IDEA TO SEND THE NEW KEY ENCRYPTED WITH THE PREVIOUS (OLD) KEY?

In general, this is not a good idea because once the old key is broken, the new key is straightforward to recover. However, sometimes the cipher is breakable with a known-plaintext attack that needs a large but feasible number of pairs. In this case it could be a good idea, since only few encrypted messages can be sent without affecting the cipher. In other words, if your cipher is only able to guarantee that the first few encrypted messages are secure, you can consider transmitting the new key within those messages and then restart the communication.

IS IT BETTER TO USE INDEPENDENT ROUND KEYS OR A KEY-SCHEDULE?

In general, it is better if the round keys are independent, that is, if each is randomly generated. To do that we need a good entropy source and an (relatively) inexpensive secure channel. However If the attacker is able to perform a (very unusual) sophisticated related-key attack, it could be a better choice to use round keys derived deterministically from a session key. But this is unrealistic.

14 DLog and Diffie-Hellman

14.1 Discrete Logarithm

Given a multiplicative group $(G, \cdot)$ and a fixed element $g \in G$, we can consider the powers of g:

$$g^0, g^1, g^2, g^3, \ldots, g^a, \ldots \quad \text{where } a \in \mathbb{N}$$

We have two cases:

- the powers are all distinct and infinite
- the powers are finite and then at a certain point they are repeated

If the group G is finite (i.e. it has a finite number of elements), only the second option can occur. From now on we always assume that G is finite.
Given any element $g \in G$, the number of distinct powers of g is called the order of g and it is denoted by $o(g) = |\{g^i | i \in \mathbb{N}\}|$. Alternatively, we can define the order of g as the smallest positive integer m such that $g^m = 1$. In this case $o(g) = m$. We use $\langle g \rangle$ to indicate the group containing all the distinct powers of g:

$$\langle g \rangle = \{g^i | i \in \mathbb{N}\}$$

. If $o(g)$ is finite, then

$$g^{o(g)} = 1$$

Example

It is easy to show that non-zero elements of $\mathbb{Z}_7$ form a multiplicative group. Let us take the powers of 2 in $\mathbb{Z}_7$.

$$\langle 2 \rangle = \{1, 2, 4, 8, 16, \ldots\} = \{1, 2, 4\}$$

Hence

$$o(2) = 3$$

and

$$2^3 = 1 \text{ in } \mathbb{Z}_7$$

Problem (DLOG)

Let $(G, \cdot)$ be a multiplicative group, and $g \in G$. Given $h = g^a$ for some (unknown) $a \in \mathbb{N}$, can we find a?
In other words, we know:

- the group G
- the element g
- the element h
- that h is obtained as some power of g .

We want to find an exponent $a \in \mathbb{N}$ such that $g^a = h$ This problem is known as **Discrete Logarithm Problem (DLOG)**.

Example

It is easy to show that non-zero elements of $\mathbb{Z}_5$ form a multiplicative group. We want to solve

$$2^a = 3 \text{ in } \mathbb{Z}_5 \backslash \{0\}$$

and the answer is

$$a = 3$$

We could take $b = 7$, and we would get

$$2^b = 2^7 = 128 = 3 \text{ in } \mathbb{Z}_5\backslash\{0\}$$

Computing a discrete logarithm can be easy or difficult, depending on the choice of the group that we use. If $(G, \cdot)$ is a group where DLOG is infeasible to solve, it is possible to describe an efficient algorithm that allows us to exchange a secret key or a seed in a secure manner on an insecure channel. This algorithm is known as the **Diffie-Hellman algorithm**.

14.2 The Diffie-Hellman algorithm: description

1. Alice and Bob agree on a public group $(G, \cdot)$ and on an element $g \in G$.
2. Alice chooses a **random** secret number $a \in \mathbb{N}$ and transmits g^a to Bob on an insecure channel
3. Bob chooses a random secret number b and transmits g^b to Alice on an insecure channel
4. Alice, knowing a and g^b, computes $(g^b)^a = g^{ba}$; Bob, knowing b and g^a, computes $(g^a)^b = g^{ab}$.
5. Alice and Bob obtain the same secret key $k = g^{ba} = g^{ab}$.

14.3 The Diffie-Hellman algorithm: Eve

We consider what Eve can do to obtain the key.

- Eve **knows**: the group $(G, \cdot)$, the element g, the powers g^a, g^b.
- Eve **does not know**: the secret exponents a, b.

With the information available to her, Eve is not able to recover the key k because she needs a and b to compute it. Eve can try to find a or b solving DLOG and obtain a from g^a or b from g^b. But **if** G is a group where solving DLOG is infeasible, then for Eve it is not possible to find either a or b in a practical way, or hence k. Alice and Bob thus get a shared secret, which they can use as session key for a block cipher or as seed for a stream cipher.

The Diffie-Hellman algorithm: example

1. Alice and Bob agree on a public group $(\mathbb{Z}_{61}\backslash\{0\}, \cdot)$ and the element $2 \in G$.
2. Alice chooses randomly $2 \in \mathbb{N}$ and transmits $2^2 = 4$ to Bob
3. Bob chooses randomly 3 and transmits $2^3 = 8$ to Alice
4. Alice and Bob have the same key $k = 2^{3 \cdot 2} = 2^{2 \cdot 3} = 64 \equiv 3$ in mod 61.

Eve knows: the group $(\mathbb{Z}_{61}\backslash\{0\}, \cdot)$, the generator 2, the powers $g^a = 4$, $g^b = 8$.
Eve does not know the exponents: 2, 3.
To recover the shared secret of Alice and Bob, $2^{2 \cdot 3} = 3$, Eve needs to find one of the secret exponents between 2 or 3. This is usually a hard problem, but in this case it is very easy for Eve to recover the secret exponents. Indeed, since $2^2 < 61$ and $2^3 < 61$, no reductions are performed in $\mathbb{Z}_{61}$ and Eve can find the secret exponents computing the usual logarithm on the real numbers:

$$log_2(4) = 2, \quad log_2(8) = 3.$$

Hence, a first trivial request of security is that g^a and g^b are greater than |G|, the size of G.

The Diffie-Hellman algorithm: example 2

Let $p = 7$. We consider as multiplicative group $G = \mathbb{Z}_7\backslash\{0\} = \{1, 2, 3, 4, 5, 6\}$ and $g = 3$ as generator.

- Alice chooses $1 \leq a \leq 6$, for example $a = 2$;
- Alice computes $g^a = 3^2 = 9 \equiv 2 \mod 7$ and sends 2 to Bob;
- Bob chooses $1 \leq b \leq 6$, for example $b = 3$;
- Bob computes $g^b = 3^3 = 27 \equiv 6 \mod 7$ to Alice, and sends 6 to Alice;
- Alice receives 6 and computes $6^a = 6^2 = 36 \equiv 1 \mod 7$;
- Bob receives 2 and computes $2^b = 2^3 = 8 \equiv 1 \mod 7$;
- Alice and Bob get the same shared secret: $1 \in \mathbb{Z}_7\backslash\{0\}$.

Eve knows the group $(\mathbb{Z}_7\backslash\{0\}, \cdot)$, the generator 3, the powers $g^a = 2$, $g^b = 6$.
Eve does not know the exponents: $a = 2$, $b = 3$.
To recover a and b, Eve tries to compute all possible powers 3^t, with $1 \leq t \leq 6$, obtaining:

$$3^1 = 3 \qquad 3^3 = 6 \qquad 3^5 = 5$$

$$3^2 = 2 \qquad 3^4 = 4 \qquad 3^6 = 1$$

concluding that $a = 2$ and $b = 3$. In this case Eve is able to recover a and b because p is very small and hence a brute force attack is feasible. To avoid a brute force attack we need the size of the group hg i to be huge.

The Diffie-Hellman algorithm: comments

The previous example shows that if p is too small, Eve might be able to break the system. Nevertheless, if p is large enough, it is infeasible for Eve to compute all distinct powers of g, unless g or p are wrongly chosen. Indeed, we have to choose g such that the powers g^t, with $1 \leq t \leq p-1$, give a lot of distinct values, so that Eve would have to compute a lot of powers to recover a secret exponent. The best choice is to take as g the primitive element of $\mathbb{Z}_p$, so that all the powers g^t, $1 \leq t \leq p-1$, are distinct.
A necessary condition for the Diffie-Hellman algorithm to be secure is that we choose a group $(G, \cdot)$ where DLOG is hard to solve. A natural question is: **can we find such groups?**
If p is a prime number, it is possible to prove that $\mathbb{Z}_p\backslash\{0\}$ is a multiplicative group. In general, in this group DLOG is hard to solve, unless p is badly chosen.

14.4 DLOG: attacks

Other algorithms exist that try to solve DLOG and are more efficient than brute force. Some of these algorithms are general, meaning that they are applicable whatever the chosen group G may be:

- Shank's algorithm
- the Pollard Rho algorithm
- the Pohlig-Hellman algorithm.

Other algorithms are more specialized and are applied to solve DLOG when $G = (\mathbb{Z}_p\backslash\{0\}, \cdot)$, for instance the Index Calculus algorithm

14.5 DLOG in practice

In cryptographic applications the most popular choices for G and g are:

1. $G = (\mathbb{Z}_p \backslash \{0\}, \cdot)$, p prime, g in $\mathbb{Z}_p \backslash \{0\}$ such that $o(g) = p - 1$;
   - It's the case we have done: you just divide your numbers by p and consider the remainders.
2. $G = (\mathbb{Z}_p \backslash \{0\}, \cdot)$, p, q primes, q divides $p - 1$, $g \in \mathbb{Z}_p \backslash \{0\}$ s.t. $o(g) = q$;
   - IGNORE
3. $G = (\mathbb{F}_{2^n} \backslash \{0\}, \cdot)$, g primitive element of $\mathbb{F}_{2^n}$;
   - IGNORE
4. $G = (E, +)$, where E is an elliptic curve on $\mathbb{Z}_p$, g is a point of E having as order a prime number, typically $o(g) = |E|/h$ with $h = 1, 2, 4$;
   - an example of groups which is very often used nowadays is formed by the points of a so-called "Elliptic Curve"; you don't need to know what is, but you must know that most web sites now use this group for their Diffie-Hellman key-agreement; you will find this method referred to as "ECDH".
5. $G = (E, +)$, where E is an elliptic curve on $\mathbb{F}_{2^n}$, g is a point of E having as order a prime number, typically $o(g) = |E|/h$ with $h = 2$ or 4;
   - IGNORE

Commonly, cases 4,5 are considered more robust, because the Index Calculus cannot be used, while cases 1,2,3 can be attacked using the appropriate form of the Index Calculus algorithm.

15 RSA

15.1 RSA algorithm: general scheme

Alice wants to send a message to Bob in a secret way.

- Bob has a public key p_k and a private key s_k
- Alice uses p_k to encrypt the message m and obtains the ciphertext c, which she sends to Bob.
- Bob decrypts c using his secret key s_k

Note that p_k is public: everybody knows it.
We show how Bob creates his public and private keys:

- Bob chooses two **random** prime numbers p and q such that
  - p, q are large
  - $log_2 p \approx log_2 q$
  - p, q are private
- Bob chooses an integer, that we call e. Thanks to his knowledge of p and q, Bob is somehow able to determine another number, called d. Bob declares his public key as (N,e), while keeping (N,d) as his private key. d and e are linked by a very special relationship, that we will see later.
- Bob computes $d \in \mathbb{Z}_{\varphi(N)}$ such that $ed \equiv 1 \ mod \ \varphi(N)$ and creates his private key $s_k = (N, d)$.
- only Bob knows the secret key $s_k = (N, d)$ while $p_k = (N, e)$ is public.

15.1.1 Encryption

- $\mathcal{P} = \mathcal{C} = \mathbb{Z}_N$, $m \in \mathcal{P}$
- Alice computes $c = m^e \ mod \ N$
- Alice transmits c to Bob

15.1.2 Decryption

Bob receives c and decrypts it, computing:

$$c^d = m^{ed} \equiv m \ mod \ N,$$

where the last congruence holds thanks to the special relationship between d and e.

15.1.3 Example

1. We choose $p = 11$ and $q = 13$
2. $N = 11 \cdot 13 = 143$ and $\varphi(N) = \varphi(143) = 10 \cdot 12 = 120$. Note that $|Z * 143| = \varphi(143) = \varphi(11) \cdot \varphi(13) = 10 \cdot 12 = 120$
3. We choose $e = 103$. Note that $gcd(e, \varphi(N)) = gcd(103, 120) = 1$
4. We compute $d = 7$, because $103 \cdot 7 = 721 \equiv 1 \ mod \ 120$

Encryption

Suppose we wish to transmit the message $m = 2$. We obtain as ciphertext $c = 63$ by reducing 2^{103} modulo 143, that is,

$$2^{103} \equiv 63 \ mod \ 143$$

Decryption

Using the private exponent $d = 7$, we get $63^7 \equiv 2 \; mod \; 143$, and so we have recovered the plaintext $m = 2$.

Suppose that Bob in the previous example, chooses a small public exponent, for instance $e = 7$. We will have:

$$c = m^e = 2^7 = 128$$

without reducing modulo N, since $128 < 143$. If this happens, the attacker, who knows e because it is public, can try to compute $\sqrt[e]{c}$, to find m. In this case she easily recovers m computing $\sqrt[7]{128} = 2$. For this reason we cannot choose a public exponent that is too small (weak key).

15.2 RSA: attacks

The attacker

- knows (N, e)
- does not know the factorization of N = pq
- does not know d

Apparently she has three ways to break RSA:

1. to factorize N
2. to compute $\varphi(N)$
3. to find d directly from the public key (N,e)

However, it is easy to prove that these three methods are essentially equivalent:

$$(1) \Leftrightarrow (2) \Leftrightarrow (3)$$

Most attacks on RSA consist in factorizing N. There are many methods to factorize a composite number. The product of two primes is the hardest case, especially if they have similar size. Some of these methods are particularly efficient if the numbers we try to factorize have special properties. Let $N = pq$. There are mainly two approaches to factorizing N.

1. To find x and y such that $x \equiv y$ mod N. An example of an algorithm using this approach is Pollard Rho.
2. To find x and y such that

$$x \not\equiv \pm y \quad mod \; N \;\; \text{and} \;\; x^2 \equiv y^2 \quad mod \; N$$

so that gcd(x+y,N) and gcd(x-y,N) are non-trivial divisors of N.
The most efficient algorithms (at the moment) used to factorize a generic integer number (meaning that its factors do not present special properties) are:

- Dixon's factorization (Dixon's Random Squares Algorithm);
- Continued fraction factorization (CFRAC);
- Quadratic Sieve;
- Shanks' square forms factorization (SQUFOF);
- General Number Field Sieve (GNFS).

16 Weak keys

The previous estimates on the time complexity required by the algorithms are evaluated considering N as an integer of medium difficulty to be factorized. In other words, we consider the **average computational complexity**. This does not mean that given any number we need the computational complexity reported by the estimation; but if we choose a random number to factorize, then the expected computational complexity is that reported by the estimates. Indeed, it is possible to find numbers that are very simple to factorize, for example because they have certain peculiarities that can be exploited by specific algorithms. Informally, a **weak key** is a key that allows a system to be easily broken. Integers that can be factorized quickly by specific algorithms are weak keys for RSA.
Example of weak keys for RSA are:

- m and e so small that $m^e < N$; in this case it is easy to compute $\sqrt[e]{c}$ recovering m;
- $|p-q| < N^{\frac{1}{4}}$; in this case there are fast algorithms able to find p e q, as for example Fermat's algorithm;
- d too small, $d < \frac{1}{3}N^{\frac{1}{4}}$; in this case the Wiener attack allows d to be recovered rapidly;
- $p-1$ or $q-1$ has only small factors; in this case N can be quickly factorized by the Pollard $p-1$ algorithm.

Unfortunately, many RSA implementations do not check if the generated keys are weak or not.

16.1 Wiener: the RSA-1024 security

If $n = \log_2(N) = 1024$, the cost for the Wiener attack is $11.1 \cdot 10^{11}$ operations.

CPU	**n.operations**	**time**
Modern laptops	2^{36} FLOPS	25.227 sec
Cluster composed by 1760 Playstation 3: 500 TFLOPS (teraFLOPS)	$5 \cdot 10^{14}$ FLOPS $= 2^{49}$ FLOPS	$2.2 \cdot 10^{-3}$ sec
Sun Yat-sen University, Cina Supercomputer Tianhe-2: 34 PFLOPS (petaFLOPS)	$34 \cdot 10^{15}$ FLOPS $= 2^{55}$ FLOPS	$3.5 \cdot 10^{-5}$ sec
2020, Cray Inc., Oak Ridge National Laboratory: 1 EFLOPS (exaflops)	10^{18} FLOPS $= 2^{60}$ FLOPS	$11.1 \cdot 10^{-7}$ sec
2030 (expected): 1 ZFLOPS (zettaFLOPS)	10^{21} FLOPS $= 2^{70}$ FLOPS	$11.1 \cdot 10^{-10}$ sec

16.2 Wiener: the RSA-2048 security

If $n = 2048$, the cost for the Wiener attack is $1.8 \cdot 10^{13}$ operations.

CPU	**n.operations**	**time**
Modern laptops	2^{36} FLOPS	410 sec
Cluster composed by 1760 Playstation 3: 500 TFLOPS (teraFLOPS)	$5 \cdot 10^{14}$ FLOPS $= 2^{49}$ FLOPS	$3.6 \cdot 10^{-2}$ sec
Sun Yat-sen University, Cina Supercomputer Tianhe-2: 34 PFLOPS (petaFLOPS)	$34 \cdot 10^{15}$ FLOPS $= 2^{55}$ FLOPS	$5.6 \cdot 10^{-4}$ sec
2020, Cray Inc., Oak Ridge National Laboratory: 1 EFLOPS (exaflops)	10^{18} FLOPS $= 2^{60}$ FLOPS	$18 \cdot 10^{-6}$ sec
2030 (expected): 1 ZFLOPS (zettaFLOPS)	10^{21} FLOPS $= 2^{70}$ FLOPS	$18 \cdot 10^{-9}$ sec

16.3 PUBLIC VS SYMMETRIC: A COMPARISON

To attack an ideal cipher with a key of k bits, the only possible attack is by brute force, which costs 2^k.
The robustness of any cipher is given by comparison with the security provided by an ideal cipher. We use k' for effective robustness (i.e. $k = k'$ if we consider an ideal cipher, but in general $k \geq k'$). For example, if a system has a space of keys containing $2^{10} = 1024$ keys, but there is an attack able to break the system using only 2^6 encryptions, we have that:

$$k = 10$$

$$k' = 6$$

NIST proposes the following comparison between the ideal security provided by a symmetric encryption and the effective security provided by DH-ECDH-RSA (IMPORTANT TABLE!).

Symmetric Key Size (bits)	RSA/DH Key Size (bits)	Elliptic Curve Key Size (bits)
80	1024	160
112	2048	224
128	3072	256
192	7680	384
256	15360	512

From the tabular it is easy to deduce that public key cryptography based on elliptic curves is the most robust: RSA and DH require bigger keys than EC to offer a protection similar to symmetric cryptography.

17 Randomness

When dealing with cryptographic primitives we often need random values. For instance:

- In symmetric cryptography:
  - keys for Shannon's perfect cipher
  - block cipher keys
  - stream cipher seeds.
- In public-key cryptography:
  - the secret exponents a and b for the DH algorithm (note that in this way ab is random, too)
  - the prime factors, p and q, of an RSA modulus N
- in digital signature (DSA)

Remark. In practical implementations of the RSA algorithm, it is difficult to choose random prime numbers. We take two random numbers and we test if such numbers are prime. Since deterministic primality tests are computationally too expensive, **probabilistic primality tests** are usually performed. Hence the numbers chosen for the RSA scheme are actually **random pseudo-primes** rather than random primes.

Ideally, the random sequences we need in cryptography should be generated from a perfect source of entropy like a coin flip or a proven perfectly random physical phenomenon. Nevertheless, perfect random sequences are extremely impractical to create. For this reason we often use pseudorandom generators rather than pure random generators (if they exist).

Definition (Pseudorandom bit generators (PRBG))

A pseudorandom bit generator is a deterministic algorithm which, given a truly random binary sequence of length k, outputs a binary sequence of length $l >> k$ which appears to be random. The input of the PRBG is called seed and the output is called pseudorandom bit sequence.
Remark. A pseudorandom bit generator can also be used as pseudorandom number generator: for a fixed bit length, it is sufficient to convert strings of bits of that length into an integer number.

A popular signature algorithm is called DSA. It uses the same groups (and so the same properties) which we saw in DH, that is, the hardness of finding a discrete logarithm modulo a prime p.

Meaning

deterministic: given as input the same seed, the PRBG always produces the same output. In particular the pseudorandom bit sequence produced by a PRBG is not random

appear: the sequence produced by a PRBG, from a statistical point of view, has almost the same properties as one expected from a perfect random sequence.

Statistical tests

To check that the pseudorandom bit sequences behave like random sequences we use standard **statistical tests**.
Typically, the random properties to test in a sequence are:

Uniformity: the sequence (and any random subsequence) must have about the same number of zeros and ones.

Scalability: any test applicable to a sequence can also be applied to subsequences extracted at random. If a sequence is random, then any extracted subsequence should also be random.

Consistency: the behavior of a generator must be consistent across starting values (seeds). It is inadequate to test a pseudo-random number generator based on the output from a single seed.

Example: The NIST Test Suite is a statistical package consisting of 15 tests that focus on several kinds of non-randomness that could exist in a sequence.

It is useful to consider that although there are many tests that can prove that a given sequence is not random, no finite number of tests can exist to prove that a sequence is random.

... and a non-statistical test

Let us consider the output sequence of a maximum-length LFSR. From a statistical point of view, the output sequences approximate a random sequence very well. Nevertheless, as already seen, such sequences can be easily predicted knowing the feedback polynomial of the LFSR, which is not difficult to recover, using the Berlekamp-Massey algorithm and having a sufficient number of output bits.

17.1 Linear complexity

The linear complexity $\ell(M)$ of a sequence of bits, M, is the length of the smallest LFSR producing the sequence.

- M maximal sequence generated by an LFSR: $\ell(M) = \log_2(M)$
- M random sequence: $\ell(M) = |M|/2$.

If a sequence has low linear complexity, it means that it can be described using a small feedback polynomial and thus it is not similar to a random sequence.

17.2 The "right" pseudorandomness

In cryptography it is not sufficient to produce sequences that are statistically similar to random sequences. We want that a PRBG outputs bit sequences that are both

- statistically similar to random sequences
- hard to predict.

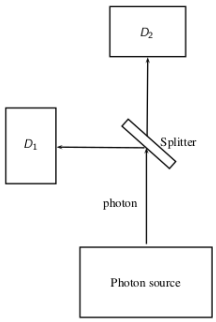
We call such PRBG's cryptographically secure.

Known examples of PRBG's that are considered cryptographically secure are constructed starting from

- block ciphers
- stream ciphers
- hash functions
- problems of number theory such as integer factorization and discrete logarithm

Physical sources of pure entropy

Even using optimal and cryptographically secure PRBG's, we still have to produce a perfect random **seed** to use as input of the PRBG. The most commonly used method to obtain a pure random seed consists of using physical sources based on phenomena that are implicitly assumed to be purely random, as for example **quantum optics**.



- A photon is sent towards a beam splitter
- the photon has the same probability to be reflected or to pass through, which is 1/2
- two detectors, D 1 and D 2 detect the direction taken by the photon
- if the photon is detected at D 1 the PRBG outputs 1
- if the photon is detected at D 2 the PRBG outputs 0

Although the process described above is in principle perfectly random, some technical factors can influence the randomness of the produced sequences, as for example:

- flaws of the splitter
- mistakes of the detector in measuring the photons
- mistakes of the device during the conversion from optical signal to electrical signal

All physical sources are affected by this kind of problems. To mitigate such issues, NIST suggests criteria and countermeasures to manage physical random sources, as for example creating probabilistic models of the physical systems.

Attention to randomness

- 270.000 sites share the same public-key of at least one other site;
- 71.052 keys are used by many different sites (thousands of sites have the same key);
- 12.720 keys RSA1024 are vulnerable and can be broken in less than a millisecond.

> *"We perform the largest ever network survey of TLS and SSH servers and present evidence that vulnerable keys are surprisingly widespread. We find that 0.75% of TLS certificates share keys due to insufficient entropy during key generation, and we suspect that another 1.70% come from the same faulty implementations and may be susceptible to compromise. Even more alarmingly, we are able to obtain RSA private keys for 0.50% of TLS hosts and 0.03% of SSH hosts, because their public keys shared non-trivial common factors due to entropy problems, and DSA private keys for 1.03% of SSH hosts, because of insufficient signature randomness[...]"*

18 Cryptographic primitives

> *"We may construct our cipher in such a way that breaking it is equivalent to (or requires at some point in the process) the solution of some problem known to be laborious."* (C. Shannon, 1949)

If you have to use an insecure channel without a previously shared secret, you are in an uncomfortable situation: you need to establish a shared secret assuming that someone is spying on the communication. In order to establish a shared secret under these critical conditions you need a mathematical problem hard to solve and a cryptographic system based on the difficulty of said problem. Number Theory offers a lot of examples of problems that are hard to solve and that can be used in cryptography. In particular, the discrete logarithm and integer factorization are the basis of many modern public-key cryptosystems. Unfortunately we do not have a complete equivalence between the difficulty of resolution of the mathematical problem and the vulnerability of the system for the common public-key cryptosystems in use. For instance, there could be an algorithm able to break RSA without factorizing the modulo. The most robust cryptosystem, at the moment, is based on DLOG over elliptic curves in a prime field. For every known public-key cryptosystem there are a lot of weak keys, which are instances easy to solve and easy to find of the mathematical problem. For example, while RSA-2048 keys require on average several millions of years to be factorized, there exist some that can be broken in few seconds using Wiener's attack or Fermat's factorization.

Private suggestions to a public cryptographer

- If at all possible, do not use public-key cryptography to encrypt messages directly, but only to exchange/agree on the key you want to use as session key for a symmetric cipher.
- Recall that public-key encryption is necessarily slower than symmetric encryption, assuming that they provide the same level of security.
- Use a good entropy source to generate the secrets involved in public-key schemes: even the best system is vulnerable if the enemy can figure out the instance of the problem he has to solve.
- Note well: public-key encryption does not provide any authentication of the users

18.1 Introduction: cryptographic primitives

A cryptographic primitive is a set of low-level algorithms that are the building blocks used to construct cryptographic protocols. Up to now we have seen three different cryptographic primitives having different roles:

- private key cryptography → to encrypt and decrypt using a secret shared key
- public key cryptography → to obtain a shared secret that can be used as symmetric key
- hash functions → to ensure the integrity of a message

We call these **principal primitives**.
Other cryptographic primitives are block ciphers, stream ciphers, random sequence generators (PRBG), DH (and ECDH), RSA.

Anything else is a "cryptographic function" (which means that it is defined using some "cryptographic primitives"). We can obtain them from the principal primitives:

- MAC → to demonstrate authenticity and integrity of a digital message;
- Digital signature → to demonstrate the authenticity of a digital message and its integrity.
- Public Key Infrastructure → to guarantee user identity in a public-key cryptographic context.
- Secret Sharing → to establish a secret shared between several users, who have to cooperate in order to recover the secret.

19 Digital signature

A digital signature is a string of bits associated to a message that allows the authentication of the signer of the message using his public key, independently from the content of the message. A digital signature is typically based on:

Hash functions + public key algorithms

A digital signature satisfies 3 properties:

- Authentication: the receiver can verify the identity of the signer.
- Non-repudiation: the signer cannot deny to have signed a message having his signature
- Integrity: the enemy cannot alter the message signed by the sender without invalidating the signature.

A digital signature scheme consists of three algorithms:

1. **Key generation algorithm.** The algorithm selects a private key, and the corresponding public key, uniformly at random from a set of possible private keys.
2. **Signing algorithm.** Given a message and a private key, this algorithm produces a signature.
3. **Verification algorithm.** Given a message, a public key and a signature, this algorithm returns "true" or "false" depending on whether the signature is valid for the message.

If an attacker is able to construct a valid message-signature pair (m, s), we say that Eve has found a forgery. The problem is that the verification algorithm returns "true" even if the message was not signed by the sender. There are different types of attack that are commonly considered:

- total break: Eve is able to determine Bob's private key, therefore she can create valid signatures on any message.
- selective forgery: Eve is able, with some non-negligible probability, to create a valid signature for a class of messages.
- existential forgery: Eve is able, with some non-negligible probability, to forge a signature for at least one message.

To avoid these types of attacks, digital signature commonly combines hash functions and public key cryptosystems.

A popular signature algorithm is called DSA. It uses the same groups (and so the same properties) which we saw in DH, that is, the hardness of finding a discrete logarithm modulo a prime p.
DSA is the main algorithm for digital signatures. It is based on the Diffie-Hellman algorithm and hence on the DLOG problem. In the standard version the algorithm requires a secure hash function. There is also a digital signature based on elliptic curves, analogous to DSA, which is called *Elliptic Curve Digital Signature Algorithm* (ECDSA). The algorithm is very similar to DSA in $(\mathbb{Z}_p \backslash \{0\}, \cdot)$, with the opportune modifications for the group $(E, +)$. Other signature algorithms in use are those based on RSA.

19.1 RSA digital signature

CREATION

Bob chooses:

PUBLIC

- Public key (N, e),
  $n = \log_2(N) + 1$

- a hash function
  Hash : $(\mathbb{F}_2)^n \to (\mathbb{F}_2)^b$, $b < n$
- $f : (\mathbb{F}_2)^b \to \mathbb{Z}_N$, f injective

PRIVATE

- Private key (N, d)
- N = pq

SIGNATURE

1. B computes
$$h = Hash(m) \in (\mathbb{F}_2)^b$$
$$m' = f(h) \in \mathbb{Z}_N$$
$$s = (m')^d \bmod N$$
2. B sends (m,s)

VERIFICATION

Alice knows N, e and f.

1. A receives (m, s)
2. A computes
$$h = Hash(m) \in (\mathbb{F}_2)^b$$
$$m' = f(h) \in \mathbb{Z}_N$$
3. A computes
$$w = (s)^e mod N$$
4. A verifies that $w \stackrel{?}{=} m'$

Demonstration: $w \stackrel{?}{=} m'$

$$w = (s^e) mod N = (m'^d)^e mod N = (m')^{de} mod N = (m')^{1+t\varphi(N)} mod N = (m') mod N$$

Since, by hypothesis, $m' \in \mathbb{Z}_N$:

$$m' = m' \; mod N$$

and

$$w = m'$$

19.2 Some comments

1. nowadays the use of RSA signature is shrinking to the protocols that formally require it (some of them in the e-payment standard) and I expect it to be phased out quickly. Indeed it has several cryptographic weaknesses that emerge when used in complex systems. It does have more sophisticated versions, such as the semantically-secure versions, but cryptographers are using more and more ECDSA
2. Indeed, ECDSA at the moment is the digital signature algorithm more used "in the wild". For example it is used in Bitcoin and many other cryptocurrencies. ECDSA needs a random source to work corectly.
3. An algorithm based again on elliptic curves, but significantly different from ECDSA, is starting to be adopted: EdDSA (used for example in Monero). It is even more mathematically sophisticated than ECDSA.

20 Secret-Sharing

The problem is: we want to share secret information with N people, called players, each of whom has a share of the secret, in such a way that the secret can be reconstructed only when a sufficient number of shares are combined together. Secret sharing is the cryptographic primitive solving this problem and other related problems. The first secret sharing schemes were independently invented by A. Shamir and G. Blakley in 1979.

Motivations and examples

- **Safeguard of secret information from loss**: secret sharing reduces the need to create backup copies of essential information, allowing arbitrarily high levels of confidentiality and reliability.
  Example: maintenance and safeguard of the cryptographic keys on a server.
- **Threshold access structure**: secret sharing allows access to the secret only once a fixed number of players share their secrets.
  Example: a company's board has to take remotely a decision between two options, "yes" or "no" (to a new deal). Each board member has a share for voting yes. If the majority of the members combine their shares, then the "yes" decision is taken. Eve cannot counterfeit a vote, because she does not have any valid share.
- **Multilevel access**: secret sharing allows access to the secret only once a group of qualified players, satisfying specific features, share the secrets.
  Example: the access to company's data is granted only to the chairman whenever he asks, the CEO whenever she asks or to the heads of department if they share their secrets.

Terminology and roles

- **Dealer**: holds the secret and distributes the shares to the players
- **Players** (Participants): hold part of the secret, distributed by the Dealer
- **Share**: the part of secret given to a player.

We call:

- $\mathcal{D}$ the dealer
- $\mathcal{P}$ the set of all players
- $\mathcal{S}$ the set of all shares

We assume that:

- $\mathcal{D} \notin \mathcal{P}$;
- the shares are secretly distributed and none of the players knows the share of other players.

In the following we show the easiest secret sharing scheme. Let us consider:

- N players, $P_1, \ldots, P_N, |\mathcal{P}| = N$
- the secret S is an integer in $\mathbb{Z}_m$, $0 \leq S \leq m-1$
- $S_i \in \mathbb{Z}_m$ is the share given to the player P_i

The goal is for **all** the players to have to pool their shares to obtain S.

- D randomly chooses S_i in $\mathcal{Z}_m$ with $1 < i < N-1$
- D computes $S_N = S - \sum_{i=1}^{N-1} S_i$ and distributes the shares

If the N players share $S_1, \ldots, S_N$ they obtain S, because

$$S_N + \sum_{i=1}^{N-1} S_i = S$$

otherwise, if only $1 \leq k \leq N-1$ players give their shares, they cannot obtain any information on S. Any time this happens the scheme is called perfect. Therefore the basic scheme is perfect.

20.1 Secret sharing: threshold scheme

Let t, w be two positive integers, $t \leq w$. A (t, w)-threshold scheme is a method to share a secret S among w participants in such a way that:

- any group of t or more players who pool their shares can recover S;
- any group of $t-1$ or fewer players who pool their shares cannot recover S (but they might recover some information on S)

A (t, w)-threshold scheme is called perfect if knowing $t-1$ or fewer shares provides no information on S, except for that publicly known.

Remark 1. The basic scheme previously presented is a (t, t)-threshold scheme.
Remark 2. In a perfect (t, w)-threshold the size of the shares is greater than or equal to the size of the secret.

20.2 Shamir's (t, w)-threshold scheme

D distributes a secret $S \in \mathbb{Z}_p$, with p prime, among $w < p$ players.

Initialization Phase

1. D sets $a_0 = S$;
2. D chooses $t-1$ random values $0 \leq a_i \leq p-1$, $1 \leq i \leq t-1$;
3. D defines the polynomial.

$$f(x) = \sum_{i=0}^{t-1} a_i x^i = S + \sum_{i=1}^{t-1} a_i x^i$$

having as constant term S (recall that $t \leq w$ and so $t < p$).

Share distribution

1. D chooses w distinct random values $1 \leq x_j \leq p-1$ with $1 \leq j \leq w$;
2. D computes $y_j = f(x_j)$ for every $1 \leq j \leq w$;
3. D gives to each player $P_j \in P$ as share the pair (x_j, y_j) for every $1 \leq j \leq w$.

Pooling of shares

t participants, $P_{i_1}, \ldots, P_{i_t}$ can recover $f(x)$ using Lagrange interpolation (theorem):

> Suppose that p is a prime number and that $x1, \ldots, x_t$ are distinct elements of $\mathbb{Z}_p$. Let $y_1, \ldots, y_t$ be elements (non necessarily distinct) of $\mathbb{Z}_p$. Then there exists a unique polynomial $f(x) \in \mathbb{Z}_p[x]$ of degree at most $t-1$, such that $f(x_i) = y_i$, $1 \leq i \leq t$.

Since $S = f(0)$, it is trivial to obtain S from f.

20.2.1 Properties

The Shamir (t,w)-threshold scheme is

- **perfect**: knowing $t-1$ or fewer shares, all values $0 \leq S \leq p-1$ are equally probable
- **ideal**: the size of one share is the size of the secret S
- **extensible to new players**: new shares (for new players) may be computed and distributed without affecting the shares of existing players
- **unconditionally secure**: its security does not rely on the difficulty of problems which are conjectured hard to solve
- and allows **varying levels of control**: providing a single player with multiple shares gives more control to that individual

Other (t,w)-threshold schemes are based on:

- geometric properties
- Chinese remainder theorem

20.3 Blakley's scheme

A secret sharing scheme exists proposed by Blakely: in general, the Blakley scheme is not ideal. Indeed, the size of the shares is t times greater than the size of the secret. Morevover, with non-zero probability we have a case where less than t players collude (secretly agree) and can recover the secret.

20.4 Mignotte's scheme

There are some secret sharing schemes based on elementary number theory. One of the most famous is due to Mignotte. However, Mignotte's scheme is not perfect, since any player P has some information on the secret, while an attacker without any share cannot recover this information. An improvement of Mignotte's idea to a perfect scheme is proposed by Asmuth and Bloom.

20.5 Generalized Secret Sharing

Suppose to have the following situation:

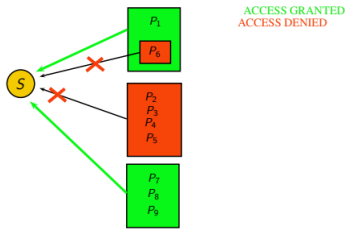
> access to important files is given to the Prime Minister and the President of the Republic. However, if necessary, three or more ministers can pool their shares in order to access the same files.

In the previous situation, the classical secret sharing scheme has to be generalized in order to guarantee different access to players who satisfy different requirements. In this case we say that an access structure is defined.

20.5.1 Access structure

An access structure is a set that contains particular subsets of players, called authorized subsets.

- players of an authorized subset can reach the secret, pooling their shares;
- players who are not in an authorized subset cannot reach the secret, even pooling their shares.



As seen for Shamir's scheme, given a (t,w)-threshold scheme it is possible to obtain an access structure, providing more than one share to the players who satisfy opportune requirements. The disadvantage of this approach is that the global size of the shares given to the players increases and, as consequence, the performance decreases.

20.6 Issues

The schemes presented up to now have drawbacks:

- if the Dealer is unfair, he can distribute shares that do not result in the right secret (once pooled);
- if some players are unfair they can pool wrong shares, causing the recovery of an incorrect secret, while obtaining information on the shares of the other players.

Those secret sharing schemes where the players can verify if the shares are correct are called **verifiable secret sharing schemes**. They can be used for electronic voting.
Examples are:

- the Feldman scheme
- the Benaloh scheme

21 Sketch on more advanced signature schemes

21.1 Threshold signatures (multisig)

In some contexts it is convenient to have a (t,w)-threshold signature scheme, that is, each of w people has a share and when t of them use their share then they can sign collectively a document. This can be done in many ways. The easiest way is to deal shares of the private key to w people using a (t,w) secret-sharing scheme, but in this way once the first signature is done the private key is recovered. There are more sophisticated ways of doing it, using i.e. secure multi-party computations or zero-knowledge proofs, that allow to sign collectively, without the need of anyone to reveal his/her share.

21.2 Ring signatures

In some contexts, anonimity is highly desired in signing documents (i.e. for unauthorized leaks). A way of achieving a high degree of anonimity is by using a scheme with "ring signatures", which we now sketch. Each user has a private key/public key pair. To sign a document, Bob uses his own private key and the public keys of an arbitrary large groups of other users (which we call the "innocents"). Alice can verify that the document has been signed by ONE of the people involved, but she cannot understand WHO signed it.

Last comment

Beware of using one elliptic curve to protect a widely-used signature systems (like a cryptocurrency). Indeed, for some specific curves there are very dangerous attacks and if Eve breaks your curve then your entire system (i.e. your cryptocurrency) will collapse. If someone breaks the Bitcoin curve she will destroy Bitcoin for good.

22 Introduction to E-Payment Systems

E-Payment Systems (EPSs) are processes and technologies enabling people to transfer money, by means of integrated hardware and software systems. EPSs are mostly used in order to enable customers to pay for some goods or services.

22.1 Electronic funds transfer at point of sale (EFT/POS)

EFT/POS is an EPS which consists in transferring money from an account to another. Usually at least one of them is a bank account. More precisely, this system is based on the use of payment cards, which are tradionally inserted in the POS, the payment terminal. Some special cards can also communicate with it in a contactless way.

In a payment card four elements are usually present:

- PAN (Primary Account Number): the credit card number, often printed on the card;
- CVV (Card Verification Value): control string printed on the card (typically on the back);
- EMV chip: integrated circuit carrying out cryptographic operations, embedded in the card with multiple physical security measures, which give it a degree of tamper resistance;
- Magnetic stripe: stripe on the back of the card, used to store the card's data.

A payment card is used for goods purchase or money withdrawal. We can distinguish 3 types:

- Credit card: allows the cardholder to obtain goods (or services) delaying the payment.
- Debit card: the money is immediately subtracted from the cardholder's (bank) account, to whom the card is directly connected.
- Prepaid card: a particular case of debit card where the cardholder's account is not a bank account, but rather an account periodically recharged by the cardholder.

22.2 Online solutions

The following points regarding payments online will be explored later:

- **Online/Mobile banking**: clients can access their account directly on the bank's website, using their own computer, and conduct the same operations they could conduct by going to their bank branch in person. In particular, they can make wire transfers to another bank account. Mobile Banking is similar, but the access is made with a mobile phone.
- **Internet Payment**: meaning nowadays the payment systems used by e-commerce websites in order to get payments from customers. We discuss two Internet payment systems:
  - **Card Not Present**: the CNP system is the traditional Internet payment system. It is still used by websites such as those of some airlines. In this system only the cardholder's details, the PAN, the CVV and the card's expiry date are needed to pay.
  - **PayPal-like**: a system in which a costumer can store her credit card(s) data in a (secure) server. Once the costumer wants to make a purchase on a website supporting the system, she will login to the system and select one of the stored credit cards.
- **Mobile payment**: payment services performed from or via a mobile device. Two examples are:
  - **Card emulation**: software architecture providing a virtual representation of a payment card. There are special POSes which can recognize mobile phones as contactless cards.
  - **Digital wallet**: e-commerce transactions can be made with the mobile phone, in a way similar to the PayPal-like systems.
- **Peer-to-peer and Micropayments**: Peer-to-peer payments are online systems permitting to a single individual to perform a financial transaction directly with another person. They include **cryptocurrencies**, such as **Bitcoin**. **Micropayments** are payments performed online and involving a tiny amount of money.

Direct data entry transactions

Namely fast paths to transmit transaction data. Divided in:

- Direct credit: payer gives instruction to her financial institution to pay funds into a payee's account;
- Direct debit: payer authorises a payee to initiate the collection of funds periodically.

22.3 Cryptography in e-payment systems

Public key cryptography

A set of algorithms using two different keys to encrypt(with the public key) and to decrypt (via the private key).

Symmetric key cryptography

A collection of algorithms that use the same key, both to encrypt a message and to decrypt it; sender and receiver have to securely produce, store and exchange it.

Digital signature

A cryptographic primitive that is used in connection with a sender, which can be a person or a device. A digital signature must satisfay three requirements: authenticity, non-repudiation and integrity. A digital signature scheme typically consists of three algorithms: a **key generation algorithm**, that selects a private key uniformly at random from a set of possible private keys and outputs the private key and a corresponding public key, a **signing algorithm** that, given a message and a private key, outputs a signature and a **signature verifying algorithm** that, given a message, a public key and a signature, checks the message's claim to authenticity.
Digital signature is one of the tools that can be used for devices; for users, other tools such as passwords and ID numbers can be used for authentication.

22.3.1 PKI

A problem in public key cryptography is to verify public keys' authenticity: if Alice wants to send Bob a message, she has to use Bob's public key. For this reason, Alice would like to be sure that the public key she is using is really Bob's. Generally speaking, a Public Key Infrastructure (PKI) is a cryptographic function that allows to:

- guarantee the identity of a user;
- associate/revocate the public key to an user.

PKI employs the following cryptographic primitives: public key cryptography and digital signature.

Certificates in PKI

A certificate is an electronic document written according to an universally accepted standard (e.g. X.509) and used to determine a univocal link between a user and her public key. It is important to guarantee that certificates have not been tampered. This is a responsibility of a CA (Certification Authority), i.e. a trusted third-party identifying the users and issuing their certificates signed with the private key of the CA.

22.3.2 TLS/SSL

They are protocols allowing secure communication between a client and a server. TLS/SSL uses:

- **PKI and certificates** to authenticate the server (and sometimes the client);
- **public key cryptography** to share a symmetric session in a secure way;
- **symmetric cryptography** to encrypt data flowing between the peers, using the session key.

23 EMV

EMV (Europay, Mastercard and Visa) is the standard used for payment cards and terminals. It uses **chip cards** (Smart cards, IC cards), which are cards endowed with an integrated circuit. The most famous implementations of EMV are:

- VIS - Visa
- M/Chip - Mastercard
- AEIPS - American Express
- CUP - China Union Pay
- D-PAS - Discover/Diners Club International

EMV requires the terminal to perform many stages of complex processing, including cryptographic authentication, to succesfully complete a transaction. We will mainly examine the cryptographic authentication. One of the tools employed in EMV is PKI.
An EMV transaction involves six actors:

- Card holder: the customer, who possesses a chip card;
- Merchant: possesses a POS (Point Of Sale) and accepts the chip cards;
- Acquirer: bank offering card acceptance service. It is usually the merchant's bank, or an entity connected to it;
- Issuer: entity which gives the card holder her chip card. It is usally the card holder's bank;
- TMS (Terminal Management System): a system which manages a POS and handles its communication with Acquirer and Issuers;
- Circuit: the entities whose applications are contained in the chip of the smartcard.

A cardholder has a contract with the Issuer which gives her the card. The card contains the cryptographic keys of the Issuer, which is owner and responsible for these keys. TMS has a contract with the Acquirer for the management of the POSes and the Acquirer has a contract with merchants to let them accept certain cards. Sometimes the TMS is not present.

23.1 Schemes for EMV transactions

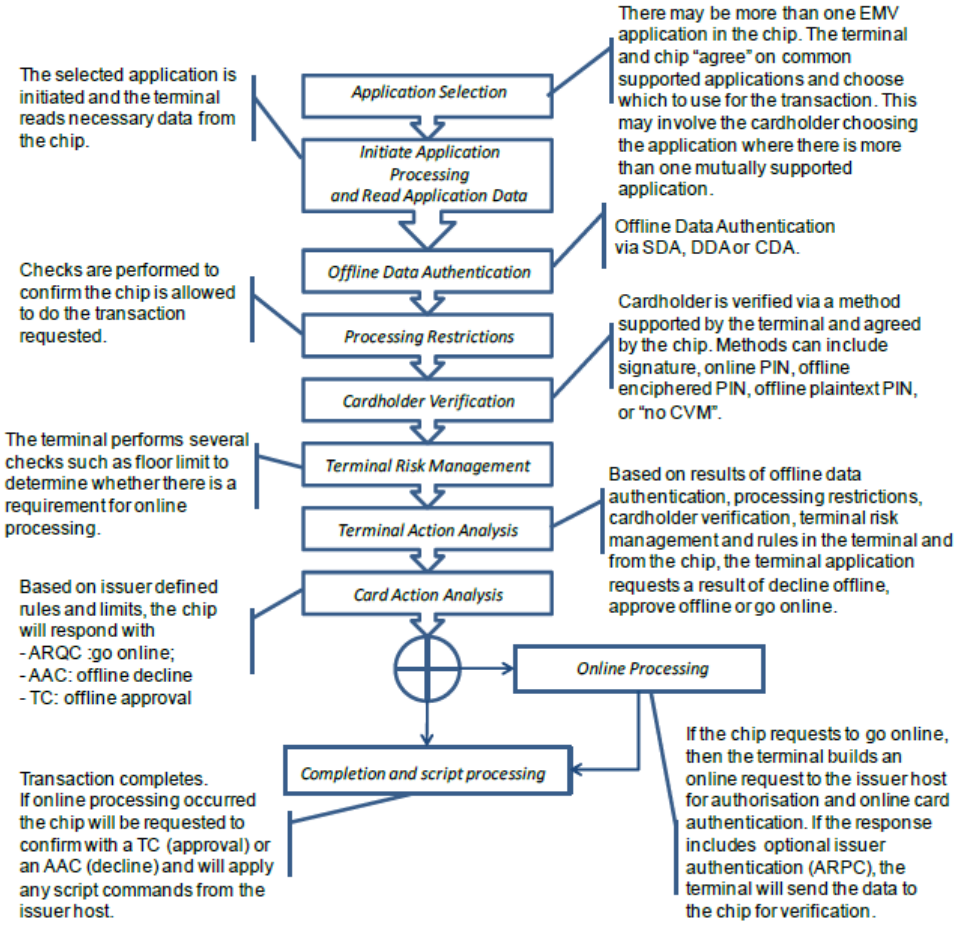
If in a transaction Issuer and Acquirer coincide, we have a three parts card scheme, sometimes called an on-us transaction.
On the other hand, if in a transaction Issuer and Acquirer do not coincide, we have a four parts card scheme, which is the most common scheme.

23.2 Card authentication

All card authentication methods are offline methods, that is, when authenticating the card, the POS does not need to communicate with the outside and it employs public key cryptography. Why offline? Because it is faster and cheaper. There is a hierarchy on existing offline authentications, with different security levels:

- a basic level called SDA (Static Data Authentication);
- a medium level called DDA (Dynamic Data Authentication);
- a high level called CDA (Combined DDA/Application Cryptogram).

23.3 EMV phases overview



23.3.1 Application selection

Identification of the circuit: the card tells the POS the supported circuits, such as Visa and Mastercard. Any circuit corresponds to one or more applications loaded in the chip: if the cards supports more than one understandable by the POS, the cardholder will choose one of these.

23.3.2 Initiate Application Processing and Read Application Data

The POS communicates with the card: *Please give me the options you need for this application and tell me which specific functions you support.* Reply by the card:

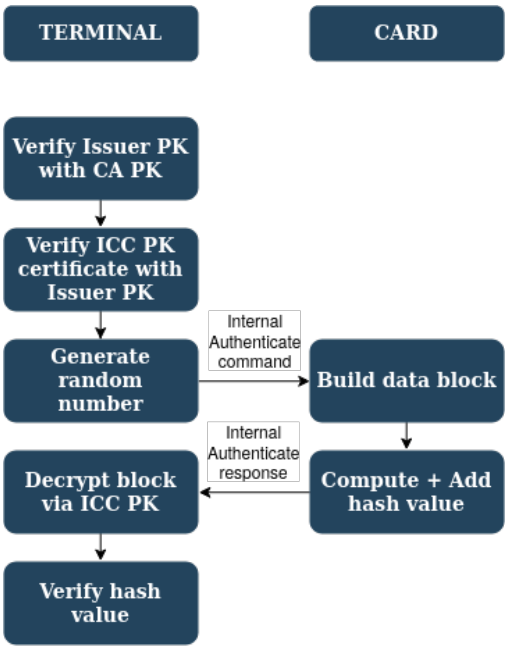
- **AIP** (Application Interchange Profile): functions which have to be performed in the transaction;
- **AFL** (Application File Locator): files needed by the terminal.

23.3.3 Offline Data Authentication

- SDA: ensures that data are signed by the Issuer, protecting from tampering but not from cloning;
- DDA: provides also application from cloning;
- CDA: combination of SDA and DDA.

DDA

The POS knows a list of public keys associated to the circuit involved in the chosen application. The card tells the POS which of these keys they will use and presents to it the Issuer's certificate. The POS verifies it, by means of the requested circuit's public key: if the verification fails the transaction is halted, otherwise the POS trusts the certificate and extracts from it the Issuer's public key.
All these previous steps are the same also in SDA, but with the next step different: the card gives the POS some static data, signed with the Issuer's private key, and the POS verifies the signature. If the signature is correct, then (SDA) card authentication is complete.
Instead, in DDA, the card tells the POS the ICC certificate, which is the card's own certificate. The POS verifies the ICC certificate, via the Issuer's public key, and so it can extract the ICC PK, which is the card's public key. Obviously, if the authentication fails, the communication is interrupted.



At this stage the POS knows that the inserted card has the public key of a valid card "NiceCard", which was issued by a valid Issuer (for that circuit). However the POS does not know if the inserted card is NiceCard! All that it knows is that the inserted card possesses some cryptographic parameters that NiceCard tells every POS. If a fake POS had obtained these values from NiceCard, the attacker might have injected them into a card, which may pretend to be NiceCard. How can the POS know the truth? By asking the card some information that can be obtained only from the NiceCard secret cryptographic parameters.
The POS sends a random number, the **UN** (Unpredictable Number), to the card, which uses it (and some other data already sent to the POS) and signs them using its private key. Encryption is done using RSA digital signature. The POS uses ICC PK to verify the signature: if it is valid, then DDA card authentication is complete.

23.3.4 Processing restrictions

In this phase, some card data read by the POS are compared to those already possessed by the POS. This is mainly a compatibility check. Now that POS is sure that the card is valid and that they can operate successfully, the POS must still be assured that the cardholder is the legitimate cardholder and not someone else that stole the card. After that, the actual transaction can take place.

23.3.5 Cardholder verification

Types of Cardholder Verification Methods (CVM):

- no CVM;
- online PIN verification (symmetric cryptography);
- offline PIN verification (PIN not encrypted);
- offline PIN verification (PIN encrypted with asymmetric cryptography);
- handwritten signature;
- a combination of handwritten signature and PIN verification.

Offline and Online PIN verification are of intereset.

Offline PIN verification

There are 2 possibilities for the ICC cryptographic parameters:

- the ICC contains specific ICC PIN encipherment key pair. The related public key is in a certificate, signed by the Issuer;
- the ICC does not possess a specific ICC PIN encipherment key pair, so a ICC key pair already used in the last stage of the DDA is reused.

The POS is linked to a device, called PIN pad. The PIN pad is temper evident and supports the entry of a 4-12 digit PIN. The PIN is entered in plaintext format on the PIN pad, formatted, possibly encrypted and sent to the POS, which creates a **PIN block**, a sequence of 16 nibbles containing the plaintext PIN. The POS sends a **Get Challenge** command to the card, that answers with an 8-bytes UN. Then the POS generates some random bytes, called Random Padding.
The POS:

- applies the PIN encipherment public key (if present) to encrypt the string

$$\text{PIN block} + \text{Header} + \text{Card UN} + \text{Random Padding}$$

- sends to the card a **Verify** command and the ciphertext.

The card uses its private key in order to decrypt the ciphertext and verifies that the plaintext is exactly as expected.

Online PIN verification

The CVM is performed directly by the Issuer host, based on the PIN typed in the PIN pad. The Issuer compares the PIN image control value, which is generated from the digited PIN and sent to the Issuer, with the PIN image stored value, which is held by the Issuer itself.
Online PINs must be encrypted using a block cipher; the longer the symmetric keys used, the better. At the moment, for online PIN encryption, only TDEA (Triple DES) and AES (Advanced Encryption Standard) can be used:

- TDEA supports different key length, but for PIN online encryption the version with a 112-bit key must be used;
- AES supports three key lengths (128, 192, 256) and any of them can be used for PIN online encryption.

The cardholder types the PIN in the PIN pad, which creates a PIN block. The PIN pad hashes and encrypts the PIN block with symmetric cryptography (TMS and PIN pad have previously exchanged a key) and sends it to the TMS, which decrypts it, re-encrypts it with a symmetric key previously exchanged with the Issuer and finally sends it to the Issuer. The Issuer decrypts the PIN block and verifies the PIN.

The actual number of symmetric keys is large, because more sophisticated cryptography algorithms and more actors are employed. What matters is that the PIN must arrive at the Issuer from the PIN pad using successive encryption/decryption phases, protected by symmetric encryption.

23.3.6 The transaction

Both the POS and the card execute some risk analysis on the transaction in object, in order to decide wether/how to perform it. There are three possible decisions:

1. Approve the transaction offline → **TC** (Transaction Certificate);
2. Decline the transaction offline → **AAC** (Application Authentication Cryptogram);
3. Process the transaction online and so involve the Issuer → **ARQC** (Authorization Request Cryptogram)

The POS generates a cryptogram, containing its decision, and sends it to the card. The card is not obliged to follow the terminal's decision, it performs its own decision-making process; even if the POS has sent a TC, the card can send and ARQC or an AAC. However, when a transaction is declined by the POS, the card cannot approve it and can only request to go online.

24 CNP and Traditional Internet Payment

With Internet payment we mean the payment systems used by e-commerce websites in order to get payments from costumers.

The **Card Not Present** system is the traditional Internet payment system: it consists of transactions made where the cardholder does not or cannot physically show the card to a merchant in order to pay. It is difficult to establish the legitimacy of the cardholder and the validity of the card, since the cardholder provides card information indirectly. The two biggest issues, often interrelated, associated with processing CNP payments are **fraud** and **chargebacks**.

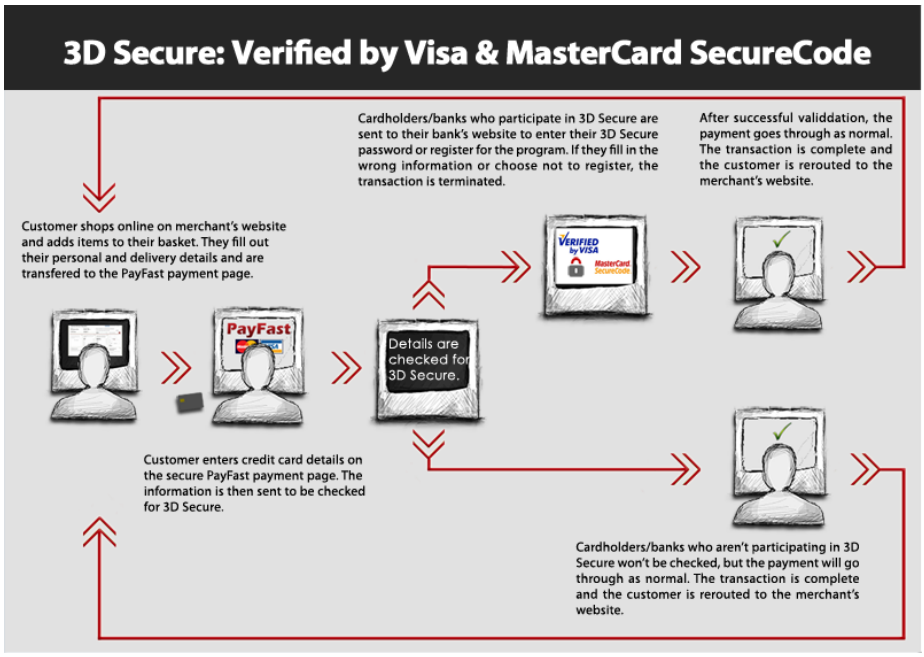
24.1 The Payment Acceptance Process

It is the verification that the card is valid and the reservation of money for the merchant; in an Internet payment it starts when a credit card is entered into the shopping cart checkout page. A payment getaway in the CNP environment is similar to a credit card terminal in a physical store: it allows the payment details to be communicated from the merchant to their merchant processor and eventually to the card Issuer. To pay, the cardholder has to digit her data and the card's PAN, CVV and expiry date. Communication is protected by TLS/SSL, but two issues remain:

- the **customer authentication**: there is no guarantee that the customer of the payment transaction is the cardholder;
- the **remote card data protection**: to improve the usability of CNP systems usually e-commerce websites store some card information such as the PAN, the cardholder name and the expiration date.

Customer authentication

In order to mitigate this problem, 3D Secure is employed, where the customer authentication is demanded to the Issuer (e.g., Verified by Visa, American Express SafeKey, MasterCard SecureCode). In principle, each issuer could use any kind of authentication method.



Whatever two-factor authentication is employed, we call that process **Strong customer authentication**; a recent document by the European Central Bank reccomends payment service providers to employ it.

Card data protection

In order to obtain the protection of the card data stored remotely on a merchant's server, different security standard requirements are employed: the most complete one is PCI DSS (29.1).

25 Online banking

In online banking, users can access their bank account directly from the bank's site, using their own computer, and conduct the same operations they could conduct going to their bank branch in person; in particular, they can make wire transfers to another bank account. They login via some credentials, such as username and password.
There are two important security aspects:

- **secure storage** of user's credentials;
- **protection** of the browser-server communication.

User credentials' protection

Usernames are stored in plaintext in the bank's server, whereas only a hash of the password is stored. Who wants to steal the user credentials has to recover the password from its hash, which is assumed to be infeasible, especially if salt is used.

Protection of the browser-server communication

To establish a private communication, the browser of the client and the bank's server start a https session, meaning that they are exploiting the TLS protocol.

25.1 Authentication

In the case of online banking the user can safely insert her username and password to login to the online platform of the bank. The authentication requiring only the knowledge of username and password is an example of Single-factor authentication (SFA). To improve the security, modern online banking systems provide a Two-factor authentication (2FA), a process that uses a combination of two different components (something that the user knows, owns or that is part of her).

25.1.1 OTP

Another possible solution is the use of a One Time Password (OTP), which is valid only for one login session or one transaction; OTPs are exempt from reply attacks. The device producing an OTP, small with a LCD display, is commonly called OTP device: the passwords produced by each device have to be unpredicatable by an attacker. Moreover, even possessing the OTPs produced by one device, it should be infeasible to obtain any information about the OTPs produced by another one. OTPs are based on CSPRNG, i.e. Cryptographically-Secure Pseudo-Random Number Generators. The sequences generated are not really random since they depend on two secret values: the **seed** (small string of bits) and the **counter**. The seed is fixed (for a given device) and the counter is increased at any use of the device. In a good PRNG function f, for each seed s and used counter c, even if an attacker knows all the $f(s,1),\dots,f(s,c)$ she cannot find $f(s,c+1)$, and if she knows s and f(s,c) she cannot find c. Not only the sequences have to be pseudorandom, also the seed distribution has to be random: if an attacker knows $s_1,\dots,s_{i-1}$ she cannot recover s_i.

There are no mathematical proofs of (pseudo)entropy of algorithms. NIST list more than 180 properties to verify entropy, plus a list of 15 tests, including **frequency**, **block frequency**, **cumulative sums**, **runs** and **longest runs of ones**.

How to generate an OTP

- **Time-synchronized**: the OTP device contains a small and accurate clock that is synchronized with the server of the bank. The time constitutes the counter, whereas each device has its own seed. The seed and the counter are given as input to a block cipher or a hash function, returning as output the OTP to display in the screen.
- **Mathematical algorithms**: each OTP is created using a mathematical function, for example iterating hash functions or block ciphers having the seed as input.

OTP devices

There are 3 different types of OTP generators used in online banking:

- **OTP devices**, the most common being:
  - RSA-like: they encrypt the seed with a block cipher;
  - Vasco-like: they use hash functions;
  - card readers: they employ the card cryptography to produce OTPs.
- **Apps**: they can permit authentication with respect to many entities.
- **SMSs**: the mobile operator involved is part of the trust chain.

A smartphone app working as OTP device is called **soft OTP**: the software implementation of the OTP generation algorithm is running on a smartphone (problems in **seed conservation**).

26 Mobile Banking

It is the same of Online Banking but the users use their mobile phone or tablet, via an app. Mobile Banking is different from Mobile Payments: communication between app and server is performed using TSL/SSL as for Online Banking.

26.1 SMS Banking

It is the oldest version of Mobile Banking. Two types of SMSs:

- Push messages: sent from the bank without customer's request (OTP or alerts);
- Pull messages: initiated by the customer who requires information or executes some operations.

Nowadays, Mobile Banking is mostly performed via ad-hoc apps, with undisclosed internal security protection mechanisms.

26.2 Security

A first layer of security in Mobile Banking is provided by TLS, then a second layer is provided by an additional encryption on top of it. A problem of Mobile Banking is the device enrolment: when the customer installs an app, she has to insert a secret in her mobile phone, to ensure that the device is actually her own and it does not belong to a potential attacker. This secret, like all meaningful data, has to be stored in a way such as it is protected (like with encryption).

Other security aspects are:

- Security of any thick-client application (running on the device). In case the device is stolen, the user should provide at least an ID/password to access the application $\rightarrow$ PIN.
- Device authentication: thanks to enrolment only authorized devices will be able to connect.
- User ID/Password authentication of bank's customer $\rightarrow$ hash.
- Encryption of the data transmitted over the air $\rightarrow$ SSL plus additional encryption layer.

Sometimes OTPs are used to prevent cyber frauds.

26.3 Certificate pinning

It is the hard-coding (=inserting code that cannot be modified without re-building the source code) of the certificate known to be used by the server in the mobile application. More precisely, the app allows only SSL connections to hosts signed with certificates stored inside the application, thus the app can ignore the device's trust storage.

27 PayPal

PayPal is a worldwide online payment system provided by an american company: it allows any business or individual with an email address to send and receive payments online. The paypal network builds on the existing financial infrastructure of bank accounts and credit cards to create a global, real-time payment solution. The **Payments API** allows to accept online and mobile payments and **Payout APIs** enable to manage payments to multiple PayPal accounts. Payments can be made with PayPal and with credit cards; a customer can accept an immediate payment, or authorize a payment and capture it later (like with subscriptions).

Using payments

The Payments API enables customers to accept PayPal and credit card payments. When creating a payment, customers set an intent (sale or authorize), and specify the transaction details. The customer can get details on completed payments (sale transactions), make full or partial refunds and similar operations.

27.1 Vault overview

The Vault API provides a secure way to store credit cards: by storing cards with PayPal, customers can avoid storing them on the e-commerce site.

- To store a customer credit card, it is necessary to specify the cards details in a call to the Vault API: a card ID is returned
- To take a payment with the card, it is necessary to specify the card ID (instead of specifying its details). If a payer ID is provided during the card storing, it is necessary to include the payer ID when charging the card.

27.2 Identity overview

If you own a website, PayPal offers an Identity API that lets your customer sign in using their PayPal credentials. The Identity API manages and verifies your customer account data for you.

- Use Log in with PayPal, a sign-on system that manages the customer log in process.
- Use seamless checkout to checkout the items in the cart.

27.3 Invoicing API

It enables the e-commerce site to create, send and manage invoices. By using it, PayPal emails the customer a link to the invoice and the customer can view the invoice on the PayPal website. Then customers can pay with PayPal, a check, a debit card or a credit card.

27.4 PCI compliance

As a payment service provider, PayPal has to comply with PCI DSS (29.1). However, for some of PayPal's products, it is the e-commerce site which is responsible for its own compliance.

27.5 Security Key

The PayPal Security Key gives a second authentication factor when logging in to a PayPal account: in addition to her password, the user must digit an OTP received in an SMS sent by the PayPal server.

27.6 Data encryption

When the customer (registers or) logs into PayPal, its server enforces a communication with TLS 1.0 or higher, ensuring that old Cipher Suites (and their weak ciphers) are not used. This might be rejected by the user's browser during the TLS handshake, making it impossible for her to log in, but it is in her security's interest.
All of her informations are protected by SSL, which ensures that they are encrypted, but no further encryption layer is employed.

28 Mobile Payment

Mobile Payments refers to payment services performed from or via a mobile device. Two of them are:

- card emulation: software architecture providing a virtual representation of a payment card. There are special POSes which can recognize the mobile phones as a contactless card.
- digital wallet: e-commerce transactions can be made with the mobile phone like with the PayPal-like systems (e.g. Google Wallet).

28.1 Google Wallet (now Google Pay)

Google Wallet allows users to store gift cards, debit cards and even PayPal credit and use it:

- to buy in shops, using the Google Wallet Card or NFC tap and pay;
- to buy online from sites that support Google Wallet;
- to send or request money to anyone in the US with an email account through Gmail or the Google Wallet app.

In order to perform the payment, Google passes a single-use virtual credit card number which can only be used for a one-time authorized transaction.

28.2 NFC payments

Near Field Communication is a Radio Frequency (RF) communication technology which enables bidirectional short-range (10 cm or less) wireless communication among electrical devices. This technology can be integrated in the SIM card, so that mobile operators can administrate NFC services on the SIM itself. It supports three methods of operation:

- **tag reader/writer**: enables NFC-enabled devices to read information stored in NFC tags embedded in labels or smart posters;
- **peer to peer**: enables two NFC-enabled devices to communicate with each other to exchange information in an adhoc fashion;
- **card emulation**: enables NFC-enabled devices such as smartphones to act like smart cards, allowing users to perform transactions such as payment or ticketing.

Only the last one will be dealt with.

In order to pay with NFC, the mobile phone needs an NFC chip and a sftware able to emulate card payments. A contactless POS wll offer the same EMV functionalities, the only difference lying in the interaction POS-card, which in this case is purely wireless, without need of any physical contact (altough a physical proximity is obviously required).

28.2.1 The payment

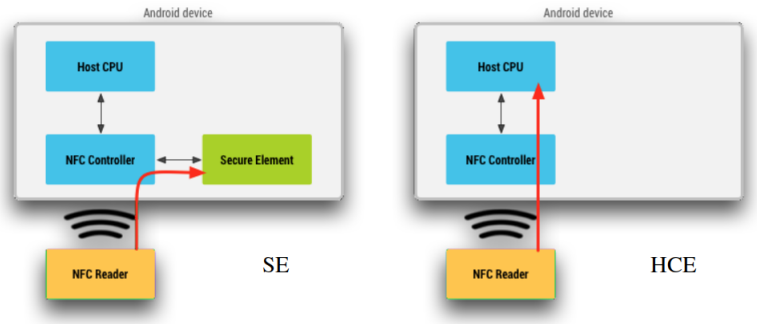
The customer holds her smartphone near the POS, in which the NFC function is enabled; usually a PIN is required for high-amount transactions. There are two important problems in NFC payments:

- how to store card data in a secure environment;
- how to protect the communication between the smartphone and the POS.

Store card data in a secure environment

Two solutions:

- **Secure Element** (SE): a small temper resistant device that facilitates the secure storage and transaction of payment and other sensitive credentials. It can be found as a chip directly in the phone's hardware, in a SIM card or in a SD card. It provides secure storage and execution environment for the payment applications, i.e. the software emulating a contactless card;
- **Host-based Card Emulation** (HCE): the host operating system and its apps gets involved with a payment transaction. All data from the reader directly go to a memory accessed by the Host CPU on which OS applications run. An application (a mobile wallet) that deals with a particular payment apllication provides for the card emulation requests and responses. The memory accessed by the host CPU is not as secure as the SE, so it does not contain the cardholder data stored inside; for example data can be moved to a hosted cloud environment, where the secure storage and secure processing takes place.



Alternatively, some OS allow SE and HCE to co-exist in the same device and offer tools to work with both.

28.3 Data exchange and authorizations

Data exchange is secured, as usual, with SSL. For authotization, OAuth 2.0 is employed: it is an open standard for authorization which provides client applications a **secure delegated access** to server resources on behalf of a resource owner.

OAuth

OAuth has been developed to work with HTTP: it allows access tokens to be issued to third-party clients by an authorization server, with the approval of the resource owner. The client then uses the access token to access the protected resources hosted by the resource server. OAuth 2.0 does not support signature, encryption, channel binding or client verification: it relies completely on TLS for some degree of confidentiality and server authentication.

28.4 Sending money with Google

Google Wallet permits to send money directly to another individual, provided that he has a Gmail account. In order to send money, the sender should turn on PIN verification. The SHA-256 hash of the PIN is stored in the device. It is also possible to choose the PIN's timeout duration: a data connection is needed to unlock the device.

29 PCI

The PCI Security Standards Council was formed by the major payment card brands: its mission is to improve payment account security. Two main standards are managed:

- the **PCI Data Security Standard** (PCI DSS);
- the **Payment Application Data Security Standard** (PA DSS).

29.1 PCI DSS

PCI DSS is a set of technical and operational requirements in order to protect the cardholder's account data; it has been developed to improve cardholder data security and encourage a broad option of consistent security measures. PCI DSS applies to all entities:

- which are **involved in payment card processing**, including merchants, processors, acquirers, issuers and service providers;
- which store, process or transmit **cardholder data** and/or **sensitive authentication data**.

Its requirements apply to organizations where account data is stored, processed or transmitted, whether they keep their data or they outsource them to third parties, which of course must respect the standard.

29.2 PA DSS

PA DSS is the PCI programme for payment applications that:

- store, process, or transmit cardholder data as part of authorization or settlement;
- are sold, licensed or distributed.

Use of a PA DSS compliant application by itself does not make an entity PCI DSS compliant, since that application must be implemented into a PCI DSS compliant environment.

29.3 PCI DSS: 12 rules

PCI DSS comprises a minimum set of requirements for protecting account data, composed of 12 general rules. Additionally, legislation or regulatory requirements may require specific protection of personal information or other data elements (for example, cardholder name).

1. Install and mantain a firewall configuration to protect cardholder data;
2. Do not use vendor-supplied defaults for passwords and other security parameters;
3. Protect stored cardholder data with encryption;
4. Encrypt transmissions of cardholder data across open, public networks;
5. Protect all systems against malware and regularly update anti-virus software or programs;
6. Develop and mantain secure systems and applications;
7. Restrict access to cardholder data by business need to know (**need to know**: access rights are granted to only the least amount of data and privileges needed to perform a job);
8. Identify and authenticate access to system components;
9. Restrict physical access to cardholder data;
10. Track and monitor all access to network resources and cardholder data;
11. Regularly test security systems and processes;

12. Maintain a policy that addresses information security for all personnel.

The **personnel** contains all the people having access to cardholder data; the **security policy** means that all personnel should be aware of the sensitivity of data and their responsibilities for protecting it.

29.3.1 Cardholder's data environment

There is an important difference between sensitive and non-sensitive data. **Cardholder Data** includes Primary Account Number (PAN), cardholder name, expiration date and service code; **Sensitive Authentication Data** includes full track data (magnetic-stripe data or equivalent on a chip), CAV2/CVC2/CVV2/CID and PINs/PIN blocks. The latter must **not** be stored after authorization, even if encrypted.

29.3.2 Network segmentation

Network segmentation of the cardholder data environment from the remainder of an entity's network is not a PCI DSS requirement; however, it is strongly recommended.

30 E-commerce

Electronic commerce is the buying and selling of products or services over electronic systems such as Internet. Merchants choosing to sell their goods and services online have a number of options to consider, for example:

- merchants may develop their own e-commerce payment software , use a third-party developed solution, or use a combination of both;
- merchants may also choose to mantain different levels of control and responsibility for managing the supporting information technology infrastructure.

There are 2 principles associated with the use of e-commerce technology and PCI DSS:

- if e-commerce technologies are used to accept payments, PCI DSS requirements apply to those technologies;
- if a merchant outsources e-commerce technologies to a third-party service provider, the merchant is still responsible to ensure PCI DSS compliance.

Two examples of third-party entities are:

- E-commerce Payments Gateway: authorizes payments for e-commerce merchants by forwarding transactions to other processors/acquirers.
- Web-hosting provider: outsourcing the website to a hosting provider.

30.1 Three-tier computing

An e-commerce infrastructure typically follows a three-tier computing model, with each layer dedicated to a specific function:

1. a presentation layer (webserver), publically accesible (not used to store confidential information);
2. a processing layer (application), that performs processing fucntions;
3. a data-storage layer.

2 and 3 handle sensitive information so they should never be publically accessible. All three are in scope for PCI DSS.

30.2 Components

Typical components of e-commerce solutions include:

- shopping cart software;
- SSL/TLS encryption;
- Network components.

30.3 Vulnerabilities

Data can be compromised for example due to web application vulnerabilities: merchants should pay attention to security when developing or selecting e-commerce software and services. The Open Web Application Security Project (OWASP) remarked the importance of secure software and the difficulty of achieving application security.

New threats are:

- **Injection flaws**;
- **Cross-site Scripting (XSS)**;
- **Cross-site Request Forgery (CSRF)**.

Old threats are:

- **Buffer Overflows**;
- **Weak authentication and/or session credentials**: vulnerable browser sessions, weak passwords, exposed protocols and services.

Secure configurations must be defined and applied to the entire e-commerce environment; the areas addressed in PCI DSS include:

- using secure encryption mechanisms in the Internet;
- protecting e-commerce components from known malware;
- using secure software development and coding practices for websites;
- limited access via **credentials**.

Reccomendations

- **Know the location of all your cardholder data**. A well designed data flow diagram will identify each system dealing with cardholder data (storing, processing and transmitting), directly or indirectly; illustrate how cardholder data is processed; illustrate where security controls are implemented.
- **If you do not need it, do not store it**. Eliminate any cardholder data that is not needed for PCI DSS; consolidate necessary cardholder data in known and manageable locations; isolate all cardholder data away from non-cardholder environments.
- **Evaluate risk associated with the selected e-commerce technology**. An agreement merchant/service provider specifies the responsibility for compliance with PCI DSS requirements for both sides.
- **Address risk associated with outsourcing to third-party service providers**.

In this last case, what merchants should do? If outsourcing

- web-hosting services: ask the provider for standard hardware and software configurations, a defined schedule for updating hardware and software patches and versions.
- data storage services: verify whether the service provider can independently manage encrypted backups and database administration.

31 Post Quantum Cryptography

- **Quantum computing**: computers based on phenomena of quantum mechanics to execute calculations;
- **Quantum cryptography**: scientific discipline that exploits properties of quantum mechanics for cryptographic goals, such as the quantum key distribution;
- **Quantum communication**: scientific discipline that exploits properties of quantum mechanics for transmitting information.

In October 2011 was opened the First academic centre devoted to quantum computing (Univ. South. California, Lockheed Martin e D-Wave Systems) and in January 2012 D-Wave claims the production of 84-qubit quantum computer. In 2013 Quantum computer D-Wave TwoTM is installed at the NASA centre. Advanced Supercomputing (NAS). In 2017 D-Wave 2000Q is designed with 2000 qubit. Unfortunately, these D-Wave computers are based on quantum annealing, rather than quantum superposition, so their purpose is not general. IBM has announced «Q System One», the first general-purpose quantum computer to be sold. This systems has 20 qubits (50 qubits are enough to compete with classical computers), it is based on quantum superposition and it is the quantum equivalent of the first computers made in the '50s and '60s. It is possible to program it and to simulate its programming.

31.1 Cryptography

Most PK cryptosystems in use are based on mathematical problems that can be solved with Shor's algorithm and a quantum computer: RSA, ElGamal, Diffie-Hellman, ECC, DSA, ECDSA, ... We need PK cryptosystems based on problems which are not solvable by Shor's algorithm:

- Based on lattices
- Based on error-correcting codes
- Based on multivariate polynomials
- Based on hash functions
- Based on elliptic curve isogenies
- Others

The USA federal institute, NIST, is actively researching new PK algorithms, quantum-resistant, in order to improve and extend its official standards which are binding for all federal entities, including:

- Recommendation (Digital Signature Standard)
- Special publications SP 800-56A Rev 2 and SP 800-56B (key establishment protocols)

32 Zero knowledge algorithm

Classic example by Micali, with $N = PQ$ (product of P and Q): given x, is there a z such that

$$x^2 \equiv z \pmod{N} \quad ?$$

1. If x exists, then z is a QR (quadratic residue) else it is a NQR. Half numbers in N are QR and the other half are NQR. If z is QR then there exist 4 **x**s so that the equation is valid.
2. You can compute efficiently these 4 **x**s and, viceversa, get P and Q if you know the 4 roots:

$$\text{To know } P, Q \iff \text{To know these 4 xs}$$

3. We don't know any efficient method to decide, given z, wether it is QR or not, unless we have P and Q.

Note:

$$QR \cdot QR = QR$$
$$NQR \cdot QR = NQR$$

if $(x, N) = 1$ and $(z, N) = 1$ (x and z coprime with N).

32.1 Micali's algorithm

There are 2 parties:

- Alice, the **prover**: knows P and Q, declares N (public) and a number z, which she claims is a QR.
- Bob, the **verifier**: does not believe Alice and challenges her.

Alice could just give Bob P and Q, so that he can compute it by himself, but she doesn't want to give him any knowledge except the fact that z is QR (don't reveal factorization of N). First, Alice has acces to a random entropy source and Bob has access to another random source (private sources); Alice calls this source and gets a number w (in $\mathbb{Z}_N$). For this random value Alice computes the square, to get $u = w^2$, which is QR, and send it to Bob. Bob calls its random source and gets a bit b, that is sent to Alice. Depending on the value of b Alice makes 2 operations:

- $b = 0$: Alice sends w.
- $b = 1$: Alice sends wx (x square root of z).

In the first case, Bob can check that the square is u (still not proving anything on z though). In the other case, Bob can square it, so that

$$w^2 x^2 = uz$$

Since w^2, x^2 and u (claimed by Alice) are QRs and since $QR \cdot QR = QR$, z has to be a QR.
So why Bob has to send b, which can have value 0? That's because Alice may be cheating and send a number that is not the square root of u: so there is a 50% probability that the b bit is 0 and Alice has to send only w. Obviously Alice can still cheat by hoping that b will be 1, so this "game" is repeated until Bob is sure that Alice is not cheating.
Also Bob could be cheating (becoming the **simulator**). Zero knowledge works so that Bob can't get extra informations other than the ones required for the computation.

This is an istance of an **Interactive Protocol**. The most specific case of IP is the case of an **Interactive Proof System**: in this case there are conditions on the probability that Bob accepts or rejects.

- **Rejection.** Bob rejects when he finds that Alice is cheating: this doesn't mean that what Alice originally said was wrong.
- **Acceptance.** Given any small number ϵ, Bob will accept the true claim with a probability of at least $1 - \epsilon$ and the false claim with a probability of at most ϵ. The ϵ usually depends on the size of the input x, so that $\epsilon \sim \frac{1}{(x)^k}$ (k is a fixed value).

Part II

APPLICATIONS

33 Passwords & Authentication

A secure system might have to track the identities of the users requesting its services. The **Authentication** is a process of verifying a user's identity. Two reasons for authenticating a user:

- The user identity is a parameter in access control decisions
- The user identity is recorded when logging security relevant events in an audit trail

It is not always **necessary** or desirable to base access control on user identities. Much stronger case for using identities in audit logs.

User authn

When logging on to a computer you enter

- username: to announce who you are
- password: to prove you are who you claim to be

This type of 'authentication' is called user authentication: the process of verifying a claimed user identity. Authentication by password is widely accepted and not too difficult to implement.

Passwords should be secrets shared between the user and the system. How do you bootstrap a system so that the password ends up in the right places, but nowhere else?

Authenticating a remote user

- Do not give the password to the caller but call back an authorized phone number from your files, e.g. from an internal company address book.
- Send passwords that are valid only for a single log-in request so that the user has to change immediately to a password not known by the sender.
- Send mail by courier with personal delivery.
- Request confirmation on a different channel to activate user account (use of SMS though is deprecated).

Resetting passwords

When setting up a new user account some delay in getting the password may be tolerated. If you have forgotten your password but are in the middle of an important task you need instant help. Procedures for resetting passwords are the same as listed previously, but now reaction should be instant. Global organisations must staff a hot desk round the clock, while, on a website, auxiliary information may authenticate a user: mother's maiden name, phone number, name of pet, ...

Guessing passwords

A basic way is brute force: try all possible combinations of valid symbols up to a certain length. Otherwise you can be smarter: search through a restricted name space. For example passwords associated with a user or popular passwords. A typical example is the dictionary attack, i.e. trying all passwords from an on-line dictionary. You cannot prevent an attacker from accidentally guessing a valid password, but you can try to reduce the probability of a password compromise.

Mitigations

First of all set a password (if there is no password, the attacker does not even have to guess it) or change the default (e.g., admin). When deciding a password, avoid guessable ones: prescribe a minimal password length, mix upper and lower case, include numerical and other non-alphabetical symbols.
In reality the most important characteristic of a password is the **length**. The longer the harder to guess, symbols can help but in a less impactful way.

33.1 Phishing and Spoofing

Identification and authentication through username and password provide unilateral authentication. A computer verifies the user's identity but the user has no guarantees about the identity of the party that has received the password. In phishing and spoofing attacks a party voluntarily sends the password over a channel, but is misled about the end point of the channel.

Spoofing

Attacker starts a program that presents a fake login screen and leaves the computer. If the next user coming to this machine enters username and password on the fake login screen, these values are captured by the program. Login is then typically aborted with a (fake) error message and the spoofing program terminates. Control returned to operating system, which now prompts the user with a genuine login request. To countermeasure this:

- Display number of failed logins, to indicate to the user that an attack has happened.
- Trusted path: guarantee that user communicates with the operating system and not with a spoofing program. For example Windows has a secure attention key CTRL+ALT+DEL for invoking the operating system logo on screen.
- Mutual authentication: user authenticated to system, system authenticated to user.

Phishing

Attacker impersonates the system to trick a user into releasing the password to the attacker. For example: a message could claim to come from a service you are using, tell you about an upgrade of the security procedures, and ask to enter username and password at the new security site that will offer stronger protection. Take care to enter your passwords only at the "right" site (but how do you know?). Another case is **social engineering**: attacker impersonates the user to trick a system operator into releasing the password to the attacker.

33.2 Passwords implementation basics

Operating system maintains a file with user names and passwords and an attacker could try to compromise the confidentiality/integrity of the password file. There are different options for protecting those files:

- cryptographic protection
- access control enforced by the operating system
- combination of cryptographic protection and access control, possibly with further measures to slow down dictionary attacks

33.3 Hash: 1-way function

A possibility to protect passwords is to use cryptographic hash functions. A 1-way function f is a function that is relatively easy to compute but hard to reverse, i.e. given an input x it is easy to compute $f(x)$, but given an output y it is hard to find x so that $y = f(x)$. Instead of the password x, the value $f(x)$ is stored in the password file; when a user logs in entering a password x', the system applies the one-way function f and compares $f(x')$ with the expected value $f(x)$.
Requirements on a hash function h:

- **Ease of computation**: given x, it is easy to compute $h(x)$
- **Compression**: h maps inputs x of arbitrary bit-length to outputs $h(x)$ of fixed bit-length n
- **One-way**: given a value y, it is computationally infeasible to find an input x so that $h(x) = y$

- **Weak collision resistance**: given an input x and $h(x)$, it is computationally infeasible to find another input x', $x \neq x'$, with $h(x) = h(x')$
- **Strong collision resistance**: it is computationally infeasible to find any two inputs x and x', $x \leq x'$, with $h(x) = h(x')$

The collision is the situation in which two inputs x and x' map to the same hash.

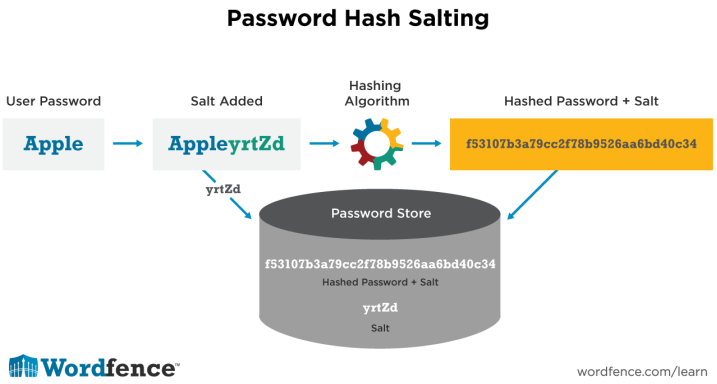
33.3.1 Implementation

- MD4: weak, it is computationally feasible to find meaningful collisions
- MD5: standard choice in Internet protocols, now broken and no longer recommended
- Secure Hash Algorithm (SHA-1): designed to operate with the US Digital
- Signature Standard (DSA); 160-bit hash value; collision attacks reported.
- RIPEMD-160: hash function frequently used by European cryptographic service providers
- SHA-256: when longer hash values are advisable

Just changing one single letter in the input string makes the resulting hashes look very different.

33.3.2 Salting

We refer to $f(x)$ as the hashed password, i.e. to the hash of x. To slow down dictionary attacks, a **salt** is appended to the password before encryption and stored with the encrypted password; if two users have the same password, they will now have different entries in the file of encrypted passwords. For example Unix uses a 12 bit salt.



Common salt implementation errors:

- Reusing the same salt for all passwords: guessing salt and use rainbow table attacks
- Using a short salt: similar attack to the one above

To use it the right way:

- Generate salt using a Cryptographically Secure Pseudo-Random Number Generator;
- Avoid the use of weak hash function implementations (e.g., MD5). Instead use PBKDF 2 (Password-Based Key Derivation Function) with a sliding computational cost, used to reduce vulnerabilities to brute force attacks;
- Rule of thumb for salt size: make it at least as long as the hash function's output.

Does the storage of the salt with the hashed password allow for a brute-force attack? No, because pre-computed rainbow table cannot be used and computing ad hoc tables with the known salt would take ages also because the salt furtherly enlarges the search space.

33.4 Handling passwords in client-server systems

In this cases don't do any security relevant operations client-side but rather server side: Hashing should be done server side, Salting should not leave the server. This does not mean that passwords travels in clear between client and server, if the appropriate protocols are used (e.g. TLS).

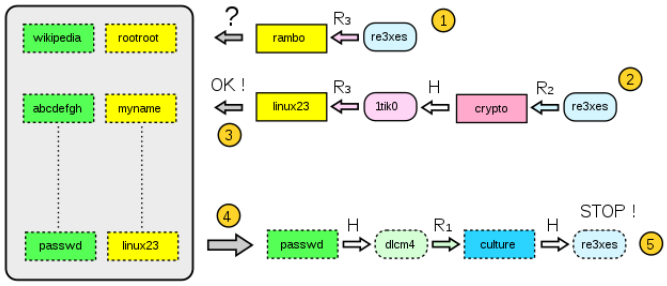
33.5 Some attacks

Look-up table attack

A look-up table is a file with pre-computed hashes stored alongside their plain text value. The savings in terms of processing time can be significant, since retrieving a value from memory is often faster than undergoing an "expensive" computation.

Rainbow table attack

A rainbow table is a precomputed table for reversing cryptographic hash functions. A rainbow table attack uses less computer processing time and more storage than a brute-force attack but more processing time and less storage than a simple lookup table with one entry per hash. This attack divides the hash into chains: a chain in a rainbow table is a list of alternating hash codes and reduction function results starting at a certain plaintext and ending at a certain hash. A rainbow table contains chains that start with an arbitrary plaintext, that is then hashed, the hash code reduced to another plaintext, that is in turn hashed, and so on. The table only stores the **starting plaintexts**, and the **final hash codes** (one chooses to end with), and so a chain "containing" millions of hashes can be represented with only a single starting plaintext, and a single finishing hash.



Dictionary attack

The dictionary attack corresponds to trying a large number of

- commonly used names as possible account names (root, webmaster, admin, mysql, oracle, guest, test, sales, staff, ...)
- commonly used passwords for the account

To mitigate it use non-obvious passwords, use allow/deny hosts capabilities of ssh (i.e. white and black lists of IP addresses from which it is / is not possible to connect) and perform log scanning; ssh servers keep a log of attempted (successful or not) connections with the originating IP address, scanning the log it is possible to detect repeated (unsuccessful) attempts that may be the indication of a dictionary attack so that further attempt connections can be disabled from a certain IP address for a given amount of time if a maximum number of tries is reached.

Credential stuffing

Attackers take millions or even billions of email addresses and corresponding cracked passwords from compromised databases and see how many of them work at other online properties. To mitigate thus, companies should extract from leaked credential lists any email addresses that correspond to their current user base, feed passwords according to standard verification process (hashing & salting) and obtain hash code C and compare stored hash code with C: if they are different, then do nothing, if they are equal, then force password reset procedure for the user.

33.6 Extensions of password authentication

33.6.1 Multi Factor Authentication (MFA)

Authentication method in which a computer user is granted access only after successfully presenting two or more factors to an authentication mechanism:

- **knowledge** (something the user and only the user knows);
- **possession** (something the user and only the user has);
- **inherence** (something the user and only the user is).

Two-factor authentication (2FA) is a type of MFA confirming users' claimed identities by using a combination of two different factors. Example : in the withdrawing of money from an ATM, only the correct combination of a bank card (something the user possesses) and a PIN (something the user knows) allows the transaction to be carried out. Other example: supplement a user-controlled password with a one-time password (OTP) or code generated or received by an authenticator (e.g., a security token or smartphone) that only the user possesses.

33.6.2 Time Based One Time Password (TOTP)

Parameters

- T0: Unix time from which to start counting time steps (default is 0)
- Tx: interval which will be used to calculate the value of the counter (default is 30 seconds)

Authenticator and authenticatee compute the TOTP value (HMAC of the counter), then the authenticator checks if the TOTP value supplied by the authenticatee matches the locally-generated TOTP value. Some authenticators allow values that should have been generated before or after the current time in order to account for slight clock skews, network latency and user delays.

Weaknesses:

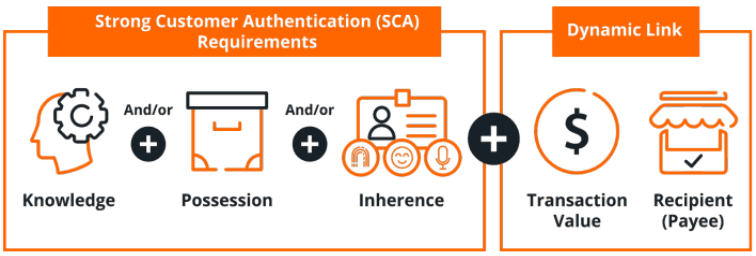
- OTP values can be phished just as passwords can, though they require attackers to proxy the credentials in real time rather than collect them later on.
- An attacker who steals the shared secret can generate new, valid TOTP values at will. This can be a particular problem if the attacker breaches a large authentication database.

An example of attack on TOTP is the one on Lockheed Marting, which used RSA for storing the secret codes.

33.6.3 Challenge Response OTP

A challenge is sent in response to access request. A legitimate user can respond to the challenge by performing a task which requires use of information only available to the user (and possibly the host). The user sends the response to the host and access is approved if response is as expected by host. The advantages are that since the challenge will be different each time, the response will be too; also it is possible to use symmetric or asymmetric crypto.

33.6.4 Revised Payment Service Directive (PSD2)



Effect = no tokens needed.

33.6.5 Use of smartphones as part of MFA

Advantages:

- No additional tokens are necessary
- Dynamically generated passcodes are safer to use than fixed (static) log-in information.

Disadvantages:

- A mobile phone is not always available - they can be lost, stolen have a dead battery or otherwise not working
- Mobile phone reception is not always available - large areas, particularly outside of towns, lack coverage
- SIM cloning gives hackers access to mobile phone connections
- Text messages to mobile phones using SMS are insecure and can be intercepted by third parties that can steal and use the token
- Modern smartphones are used both for receiving email and SMS. So if the phone is lost or stolen and is not protected by a password or biometric, all accounts for which the email is the key can be hacked as the phone can receive the second factor

33.6.6 Passwordless authentication

Example: FIDO (`www.fidoalliance.org/how-fido-works/`)
The FIDO protocols use standard public key cryptography techniques to provide stronger authentication. During registration with an online service, the user's client device creates a new key pair. It retains the private key and registers the public key with the online service. Authentication is done by the client device proving possession of the private key to the service by signing a challenge. The client's private keys can be used only after they are unlocked locally on the device by the user. The local unlock is accomplished by a user–friendly and secure action such as swiping a finger, entering a PIN, speaking into a microphone, inserting a second–factor device or pressing a button.

33.6.7 Assurance Level (NIST)

Identity:

- Self-asserted; no requirement to link the applicant to a specific real-life identity.
- Evidence supports the real-world existence of the claimed identity; either remote or physically present identity proofing.
- Physical presence is required for identity proofing. Identifying attributes must be verified by an authorized and trained representatives.

Authentication:

- Provides some assurance that the claimant controls the authenticator; requires at least single-factor authentication.
- Provides high confidence that the claimant controls authenticators; two different authentication factors are required; approved cryptographic techniques are required
- Provides very high confidence that the claimant controls the authenticator; authentication based on proof of possession of a key through a cryptographic protocol; requires a "hard" cryptographic authenticator

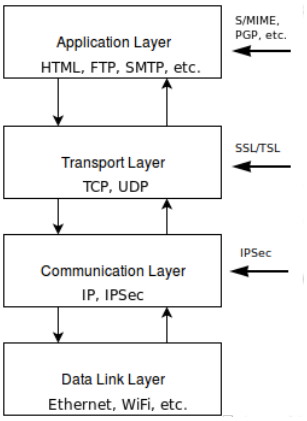
34 Public Key Infrastructure

The goals of information security are preserving integrity and/or confidentiality of data from sending point to the receiving point in a network and providing for the receiver to be certain that the sender is actually the entity that the information was received from. These require:

- **Authentication**: when information is received from a source, the source is indeed as alleged in the information and the information was not altered along the way (integrity)
- **Confidentiality**: information is safe from being eaves-dropped on during its transit
- **Choosing best parameters and key management**: two endpoints of a communication link may possess different computational capabilities and may not have access to exactly the same set of security algorithms

Authentication, confidentiality, and key management can be implemented at different layers of the communication stack:

- **Network layer**: covers all applications running over the network. Largest application of IPSec is in Virtual Private Networks (VPN = overlay network that allows a group of hosts that may be widely scattered in the internet to act as if they were in a single LAN). Also known as the "*Message level security*".
- **Transport layer**: provides Session Layer security in the OSI model of the internet protocols; plays a central role in the security and privacy needed for web commerce to work. Example: before your laptop uploads your credit card information to on-line shopping, it must make certain that the remote host is indeed what it claims to be. Also known as the "*Session level security*" (session = enduring association between a client and a server consisting of multiple connections).
- **Application layer**: S/MIME, PGP (Pretty Good Privacy), OpenPGP. Initially developed for securing emails, now used for securing information at rest (to mitigate malwares). Also known as "*Email level security*"



A Public Key Infrastructure (PKI) is the cryptographic primitive that:

- binds public keys with their respective users and, if necessary, revokes specific public keys
- guarantees user identification

The main components of PKI are:

- certificates
- certification authorities (CA)
- registration authorities (RA)

34.1 Public key cryptography and key distribution

Public Key Cryptography (PKC), or asymmetric cryptography, is a cryptographic system that uses pairs of keys: public keys which may be disseminated widely, and private keys which are known only to the owner. Effective security only requires keeping the private key private; the public key can be openly distributed without compromising security. In this system, any person can encrypt a message using the receiver's public key, but that encrypted message can only be decrypted with the receiver's private key. This can be done also in the opposite way to provide authentication: a sender can combine a message with a private key to create a short digital signature on the message. Anyone with the sender's corresponding public key can combine the same message and the supposed digital signature associated with it to verify whether the signature was valid, i.e. made by the owner of the corresponding private key.

34.1.1 Pros and Cons

Pros:

- Private key is only known by the owner (less risk)
- Integrity and Confidentiality by encrypting with Receiver's Public key
- Non-Repudiation by encrypting with Sender's Private key

Cons:

- Algorithms are 2-3 orders of magnitude slower than those for symmetric encryption: so it is typically used in an initial phase of communication and then secrets keys are generated to deal with encryptions
- How are Public keys made available to the other people?
- There is still a problem of authentication: who ensures that the owner of a key pair is really the person whose real life name is "Alice" (sender)?

With symmetric encryption, it is needed the exchange of $N \cdot (N-1)/2$ key pairs (at least) where N is the number of users; if the internet users are 4,536,248,808 the number of key pairs is around $1.02888 \cdot 10^{19}$.

RSA Factoring challenge

For each RSA number n, there exists prime numbers p and q such that $n = p \times q$. Problem: find p and q primes, given only n. Started in 1991, ended in 2007. The practical difficulty of factoring large integers and the cracking of RSA keys used in cryptography (Moore's law) make RSA looks outdated. Given Moore's Law (the number of transistors in a dense integrated circuit doubles about every two year), when will that key size be inadequate? A better encryption method right now is Elliptic Curve Cryptography (ECC).

34.1.2 Public key distribution

How to find out the right public key of an entity? I.e. how to implement the key ring?
Possible answers:

- In person public key exchange: simple and secure, but not very convenient;
- Public announcement via newsgroups, web pages, etc: forgeries are possible and it is hard to spot liars;
- Diffie-Hellman solution, a public-key directory (similar to phone number directory): should be always available, must have no authorized modification, secure and authentication communication between directory and users;
- Trusted public key authorities that know public keys of entities and distribute them on-demand: on-line protocol;
- Certificates: binding between public keys and their owners.

34.2 Certificates

To bind a public key to the identity of a user in an unambiguous manner, a PKI employs **certificates**. Certificates are the building blocks of PKIs, digital documents that typically contain a public key, data referring to the entity that emitted the certificate (CA), data referring to the certificate owner (for example: name, country, etc.), a validity period, encryption algorithms, digital signatures and key-exchange algorithms supported. Different fields can be contained in a certificate, depending on the international protocols that define their format.
When Alice wants to send a message to Bob:

- she asks Bob to send his certificate
- Bob sends his certificate
- Alice opens the certificate, finding Bob's public key, and she uses it to send her message to Bob.

However, even using the certificates, the problem still remains: Alice cannot be sure that the certificate really belongs to Bob. To solve the problem a PKI uses trusted third parties, called Certification Authorities (CA), that guarantee the identity of users and control the issuing of certificates. Once the identity of users requiring a public key has been checked, a CA issues a certificate signed with its own private key, to validate the public key of the user. **The CAs sign their own certificates**.

Certificates issues

There are issues with CAs:

- **CA certification policies**: they describe the methodology of certificate issuance. There are different types of identity control practices:
  - loose control: only email address
  - tight control: apply in person and submit picture IDs and/or hard documentation
- **Trust**: verifiers must trust CAs, CAs need not trust the certified entities and a certified entity needs not to trust its CA, unless it is not the verifier. Trust is based on how correct is the certificate information, so it is related to the certification policies.

Certificate types:

- ID certificates (for authentication);
- authorization certificates: no identity, binding between public key and authorization info.

Storage and distribution

Certificate **storage** and **distribution** can be done along with a signed message, with distributed directories or with centralized databases.

Revocation

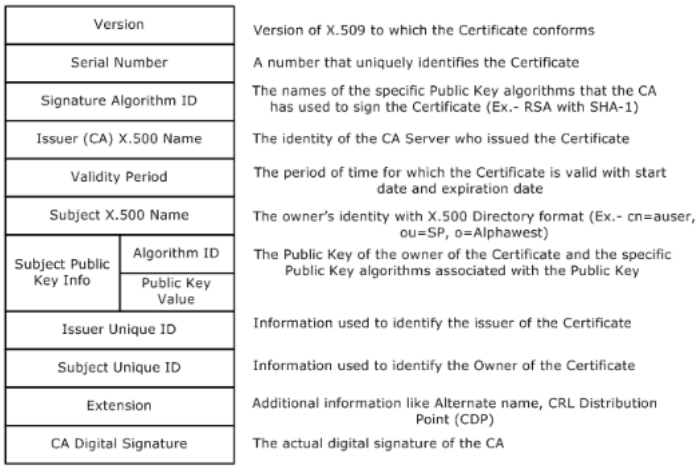
Certificate also have lifetimes, but they may be **revoked** before the expiration time because of certificate holder key compromise/lost, CA key compromise and end of contract (e.g. certificates for employees). The **Certificate Revocation Lists** (CRLs) hold the list of certificates that are not expired but revoked.

Relationship with real world ID docs

Is a certificate an "electronic identity"? A certificate is a binding between an identity and a key, not a binding between an identity and a real person; one must submit its certificate to identify itself, but submission is not sufficient, the key must be used in a protocol. Anyone can submit someone else's certificate. Certificates are not picture IDs, so, what is the real world analogy for certificates? Endorsed document/card that serves as a binding between the identity and signature, for example "credit-cards".

X.509 standard

There is a ITU (International Telecommunication Union) standard for certificates, X.509: ISO 9495-2 is the equivalent ISO standard. It defines certificate structure, not PKI, and the authentication protocols. It is used for identity certificates. Supports both hierarchical model and cross certificates. End users cannot be CAs.



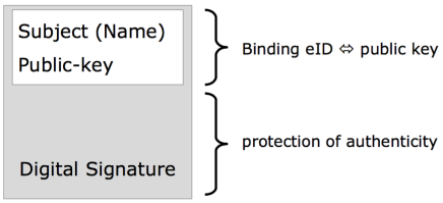
34.3 Public Key Infrastructure

The goals of a PKI are to assure that the public key is **available**, **authentic** and **valid**. It also has to enforce **security** and **interoperability**.

34.3.1 Authenticate public keys

- Bind public key to electronic identity
- Seal the binding
- Answer for the binding

Public key certificates are data structures that bind public key values to subjects. The binding is asserted by having a trusted CA digitally sign each certificate.



34.3.2 Security of key pairs

- Select suitable algorithms and key sizes
- Monitor possible security threads and react adequately
- Provide suitable means to generate key pairs
- Provide suitable formats and media to store private keys
- Provide suitable means of delivering private keys

Security also depends on **personal security environments** (e.g., a mobile computing device with a smart-card reader for creating digital signatures).

OpenSSL

With OpenSSL it is possible to generate a 2048 bit RSA Key and export the RSA Public Key to a file.

Protect your keys

To protect your keys you must keep the private key backed up and secret, while the public key can be distributed anywhere or embedded in web application scripts or other scripts. If the private key goes away, the data encrypted to it is gone; keeping a printed copy of the key material in a sealed envelope in a bank safety deposit box is a good way to protect important keys against loss due to fire or hard drive failure.

34.3.3 PKI

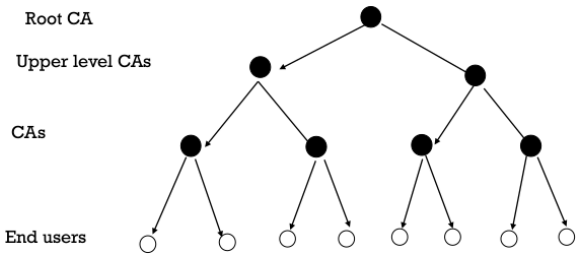
A PKI is an arrangement that binds public keys with respective identities of entities (like people and organizations). An entity must be uniquely identifiable within each CA domain on the basis of information about that entity. The binding is established through a process of registration and issuance of certificates at and by a **Certificate Authority** (CA): this can be an automated or manual process, depending on the level of assurance. The PKI role that assures valid and correct registration is called a **Registration Authority** (RA); RA is responsible for accepting requests for digital certificates and authenticating the entity making the request. A third-party **Validation Authority** (VA) can provide an entity information on behalf of the CA. The **certificate distribution** system is a repository for certificates (e.g., LDAP) plus the **Certificate Revocation List** (CRL).
PKI is a complete system and defined mechanisms for certificates

- certificate issuance
- certificate revocation
- certificate storage
- certificate distribution

Their business practice consists in issueing certificates and make money. Several CAs are also necessary due to political, geographical and trust reasons. An entity assumes CA will establish and maintain accurate binding (secure the binding, CA's keys binding, security, etc...) of attributes and public keys.
In a PKI the CAs can be organized in different structures, called trust models, that also limit/-control trust in a given environment:

- hierarchical
- cross certificates
- hybrid

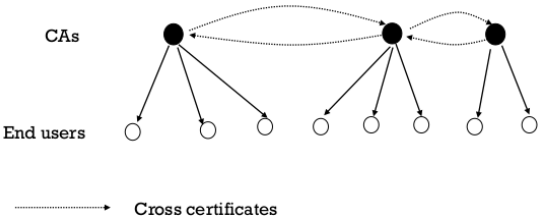
Hierarchical PKI



It has a tree architecture, with a single root CA and a number of subordinate CA's (recursively); parent certifies children and leaves are non-CA (end-) entities. Typically CA either certifies other CA's or end-entities, but not both. Everyone has root CA PK.

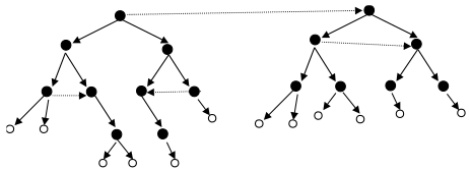
Example: u_1, to validate u_8, goes to Root CA and uses its public key to validate its certificate. u_1 validates the certificate of CA_6 using the public key of Root CA, which originally signed it with its private key. u_1 extracts CA_6 public key from the certificate and validates the certificate of u_8, which signed this certificate with its own private key.

Cross certificate PKI



Cross-certification extends third-party trust relationships between CA domains. Certificates are for CAs (not end-entities). A cross-certificate is called an "inter-domain cross-certificate" if the subject and issuer CAs belong to different administrative domains; it is called an "intra-domain cross-certificate" otherwise. There are one or two directions (CA1 certifies CA2 and/or CA2 certifies CA1). Control is implemented because cross-certificate limits trust (name, policy, path length, etc. constraints).

Hybrid PKI



A hybrid architecture arises as a result of interconnecting independent, pre-existing PKIs from different organizational domains. The process of interconnecting the peer root CAs is commonluy know as **cross-certification**, although the term **PKI networking** is growing in use. It is fully-meshed (all CAs are cross-certified). However, all trust models that use cross-certification present the following drawbacks:

- increased complexity of the algorithms for certification path construction;
- possible creation of unwanted trust paths through transitive trust;
- decrease of the security level when a high-assurance PKI with restrictive operating policies is cross-certified with less restrictive policies.

Also the hybrid model serves well to link a small number of hierarchies, but does not scale well. Due to this problem, a Hub & spokes configuration, also known as the **Bridge CA** model, was developed. The BCA acts as a central point to establish trust paths among different PKIs, rather than act as the root of certification paths. Each CA can operate independently and under different certification policies, holding its CA PK. In this environment, the number of relationships grows only linearly with the number of PKIs.

34.3.4 Certificate paths

Verifier must know public key of the first CA, other public keys are found out one by one. All CAs on the path must be trusted by the verifier.

Certificate path and processing

Alice "trusts" CA1: Alice has CA1's PK in its browser. CA1's PK is called "trust anchor" and depends on the model. CA1 certifies CA2; CA2 certifies CA3; CA3 certifies Bob. So Alice "trusts" Bob. Alice associates PK in Bob's certificate with Bob.

- Path construction: aggregation of necessary certificates;
- Path validation: checking the certificates and the keys (includes all steps of certificate validation).

Path construction is just the shortest path graph algorithm (not so simple because graph is not known and the edges, standing for the certificates, need to be queeried); once Path Construction is done, Path Validation is straight-forward.

34.4 A critique of PKI - NOT TO STUDY

Who do we trust and for what?

A CA is seen as trusted, but for what? Many CAs disclaim all liability and any meaning to the certificate; this does not mean one can trust a certificate for a given application.

An example for ID Cards: the CA is trusted for identifying persons (claim certificates are for that purpose); this means one can trust a certificate for authentication, not for authorization. Exceptions are for Identity Based Access Control (IBAC) and for when authentication is complemented with an infrastructure for Attribute Based Access Control (ABAC).

Who is using my key?

The biggest risk in PKI is with private key: how to protect it? When stored in a smart-card, how attack resistant is the card? Assuming the card is attack resistant, can a malicious driving app/device get the trustworthy card to sign something one did not intend to sign? This implies non-repudiation: this is ok for cryptographic reasons (no signature forgery), not ok for legal reasons (if someone uses your private key, then you are not allowed to repudiate the signature).

An example for ID Cards: the key is protected by PIN, no idea how "attack resistant" it is. A malicious mobile app can induce to sign something one did not intend to sign but there are mitigations: always half a PIN but the first time.

How secure is the verifying computer?

Verifying computer (the one using the certificate) shall be secure. Certificate verification uses only public keys, thus there are no private keys to protect but one or more root public keys are used. If attackers add their own public key to root keys, then they can issue their own certificates that match legitimate certificates in every other field except for the public key.

For the case of ID Cards: is the Viminale server secure? Is there a need of a more careful security assessment?

Which John Robinson is he?

Certificates associate a public key with a name: how useful is this association? Example: in their seminal paper, Diffie & Hellman introduced a modified telephone directory in which one can read name, address, and public key. This might have worked with the Stanford CS Dept phone directory in 1976 but it certainly does not scale to the Internet.

Back to ID Cards, certificates associate a public key with a "person profile" that includes name, surname, address,

..., and CF. Additional information is provided to comply with ICAO Doc 9303 standard that should make that name unique in the world.

Is the CA an authority

A CA may be an authority on making certificates but is it an authority on what the certificates contain? Example: in SSL/TLS, certificates contain the keyholder (corporate) and DNS name for the server. There exist authorities on DNS name assignements, but none of the SSL/TLS CAs in popular browsers is such an authority.

The combination of IPZS, Viminale, and municipalities is an authority on both making certificates and identifying persons.

Is the user part of the security design?

Does the application using certificates take users into account? Example: a user makes a decision whether to interact with a SSL/TLS-protected web page based on what is displayed on that page. Certificate is not displayed and does not have a relation to what is displayed; SSL/TLS security cannot control or react to the content of the web page, only its DNS address.

The certificate of the server is not shown to the user who cannot verify anything. The check on the server certificate is performed at the cryptographic level only. See also the issue of certificate management in mobile applications verified by TLSAssistant: would it be meaningful to involve the user in checking server certificates somehow?

Was it one CA or a CA + a RA?

In response to the lack of authority on the content of certificates, some CAs have created a 2-part certification structure:

- Registration Authority (RA) for content
- CA for certificate management

RA+CA model has some security issues: CA may forge certificates despite contract binding it not to do so.

How did the CA identify the certificate holder?

CA needs to identify the applicant before issuing a certificate. Once the CA identified the applicant somehow, how did it verify that the applicant control the private key corresponding to the public key being certified?

Identification is performed in a standard and well-established way by the combination of Viminale and municipalities. The verification that the applicant control the private key corresponding to the public key being certified is side-stepped because it is the CA that generates the pair of keys.

How secure are the certificate practices?

Certificates must be used properly to ensure security:

- How is key lifetime computed?
- Are CRLs supported?
- Can revocation be retroactive?
- How long are the generated public keys?
- Does the proper use of the certificates require user actions?

For the ID, the answers to most of the questions can be found in the ICO standard Doc 9303 on PKI. For the last question: to use the certificate in the card, the user must enter the PIN. No action is required to handle the certificate from the server (see comments to risk 6).

Why are we using the CA process, anyway?

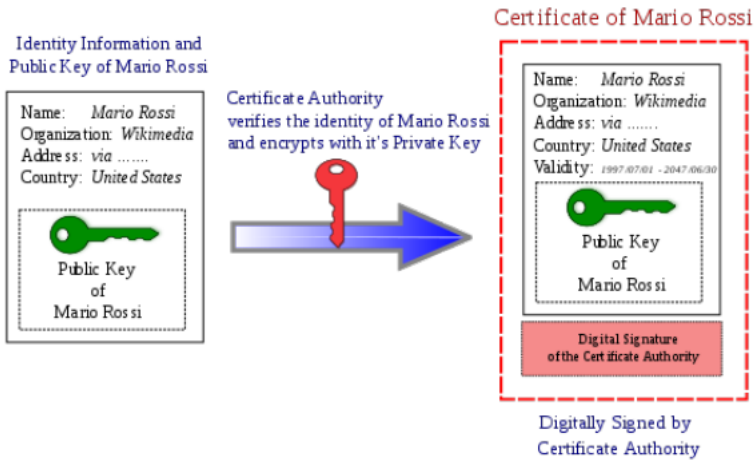
Customer: how do we add SSO on top of PKI?

PKI vendor: you don't! That requires massive changes in the underlying system.

For ID Cards, when used as a 2nd factor authentication, it can be blended with SSO in a meaningful way.

34.5 ACME and Let's Encrypt

34.5.1 Obtaining a certificate



1. User-A generates a Public and Private key-pair or is assigned a key-pair by some authority in their organization
2. User-A first requests the certificate of the CA Server
3. The CA responds with its Certificate including its Public Key and its Digital Signature signed using its Private Key
4. User-A gathers all information required by the CA Server to obtain its certificate; e.g., User-A email address, fingerprints, etc. that the CA needs to be certain that User-A claims to be who she is
5. User-A sends a certificate request to the CA consisting of her Public Key and additional information. The certificate request is signed by CA's Public Key.
6. CA gets the certificate request, verifies User-A's identity and generates a certificate for User-A, binding her identity and her Public Key. The signature of CA verifies the authenticity of the Certificate
7. CA issues the certificate to User-A.

34.5.2 Creation of certificates

Who are CAs? For profit companies that get paid for creating certificates after vetting clients. Vetting process may be manual or partially automated. For certain types of certificates (e.g., EV - Extended Validation), vetting is manual and allows for verifying the official identity. This is important for certain services, e.g., financial or government; for other types of certificates (e.g., DV - Domain Validation), vetting can be automated as it is only needed that holders control the domain name. This is the case for the vast majority of certificates currently in use: for instance, certain CAs may require to answer emails sent to contact addresses of holders.

34.5.3 How to create a certificate

First, create a Certificate Signing Request (CSR), then, submit the CSR to the selected CA. After receiving the CSR, the CA must perform several checks with different levels of accuracy. For instance, CA uses *whois* to obtain the email address of the contact persons, send a message containing a challenge; the client must reply to prove that he/she can access the address and control the domain. At this point, the CA creates a certificate that is sent to the client; the certificate must be then installed in the server (e.g., https). This complex process can be difficult for newcomers: **ACME** aims to normalize it and make it fully automated.

34.5.4 Automated Certificate Management Environment(ACME)

CA runs an ACME server that waits for CSRs. An ACME client:

- generates a CSR
- connects to the ACME server
- asks for a signed certificate for a given name

The ACME server generates a challenge for the client which can prove to control the domain name for which the certificate has been asked; there exist several challenges but the most common consists of the ACME server to ask the client to create a file with a given name and given content in a certain position of the web site. If successful, the ACME server concludes that the client controls the domain name and signs the certificate that is sent to the client after submission of the CSR.

34.5.5 Let's Encrypt

It is a free, automated, and open CA, run for the public's benefit, provided by ISRG - Internet Security Research Group (California based Non-Profit) and sponsored by Mozilla, Akamai, Cisco, EFF, OVH, Facebook, Chrome, etc. Its mission is to reduce technological and financial barriers to secure Internet communication, by encrypting all communications. The ACME protocol is used to generate certificates, which can only be Domain Validation Certificates, not Extended Validation.

34.5.6 Certificate Transparency (CT)

CT is an Internet security standard and open source framework for monitoring and auditing digital certificates. The standard creates a system of public logs that seek to eventually record all certificates issued by publicly trusted certificate authorities, allowing efficient identification of mistakenly or maliciously issued certificates. CT was developed as a result of a security incident with a CA: in 2011, a reseller of the certificate authority Comodo was attacked and the certificate authority DigiNotar was compromised, calling attention to existing flaws in the certificate authority ecosystem and accelerating work on various mechanisms to prevent or monitor unauthorized certificate issuance. Ben Laurie, Adam Langley and Emilia Kasper began work on an open source framework to combat these issues the same year. One of the problems with digital certificate management is that fraudulent certificates take a long time to be spotted, reported and revoked by the browser vendors. CT would help by making it impossible for a certificate to be issued for a domain without the domain owner knowing; it does not require side channel communication to validate certificates as do some competing technologies such as Online Certificate Status Protocol (OCSP). CT also operates without the need to trust a third, it depends on verifiable Certificate Transparency logs. A log appends new certificates to an ever-growing Merkle hash tree.

Example of Facebook and Let's Encrypt

In 2016, Facebook CT monitoring service allowed to spot that the Let's Encrypt CA issued two TLS certificates for multiple fb.com subdomains. These two certificates raised red flags for Facebook team because they were not issued by their primary CA vendor, they were not authorized by their security team and they were shared with multiple domains that they did not own or control. Facebook determined that these certificates were requested by the hosting vendor managing these domains for several of tehir microsites. Vendor had authorization from another Facebook team to use Let's Encrypt, but that detail was not communicated to the security team. The investigation was completed in hours and certificates were revoked. They found no indications that these certificates were ever controlled by unauthorized parties, and were able to respond before they had been deployed on the production hosts. Like many large organizations, Facebook uses microsites for things like marketing and special events. These sites are built and hosted separately from the main product platforms. While outsourcing these sites may impact the control over their security posture, it also isolates them from the core systems and information about people on Facebook. CT monitoring allows to track these sites even if the direct management is delegated to another party.

34.6 Applications of PKI

34.6.1 DNSSEC

It is a security extension to DNS, not X.509 based but hierarchical (uses existing DNS topology) and distributed. It Provides;

- authentication of domain information and storage
- distribution of certificates

34.6.2 SSL/TLS

Security layer over TCP/IP, mostly for HTTP connections; enables encrypted and authenticated sessions between web servers and web browsers (clients). It is not a perfect solution, but a convenient one. In TLS web servers **must have** a certificate, client certificate is optional. CA certificates are embedded in browsers; you trust them (by default), because browser company says so (the worst, but the most practical...).

By using SSL in HTTP connections, we can:

- make sure about the server's name (assuming server's CA is trusted) = **authentication**
- make sure that nobody can see the traffic between client and server = **confidentiality**

However, we can NOT:

- provide perfect privacy: server sees all information that client provides important in e-payment (merchant sees the the card number and name);
- provide non-repudiation: both parties knows the session key in e-payment (charge-back cost for merchant's).

34.6.3 Verification of a certificate in TLS

Server is authenticated by the client as follows (roughly):

1. server has the private key corresponding to the public key in the certificate (and thus can sign messages);
2. Certificate is not expired;
3. Certificate is signed by a CA known to the client (client knows the public key of the CA);
4. The name indicated by the client corresponds to one of the names in the certificate.

After this authentication is over, TLS is able to guarantee integrity and confidentiality of communication. Authentication is NOT trust: it only means that server has the name indicated and that malicious third parties cannot hear/modify exchanged messages.

34.6.4 S/MIME

- MIME (Multipurpose Internet Mail Extensions): content type, almost any of information can appear in an email message
- S/MIME: Secure MIME: new content types, like signature, encrypted data

It is a IETF standard way for email encryption and signing. It has industry support for commercial reasons. Not a standalone software, a system that is to be supported by email clients.
The general functionality is the digital signature. The hash of a message is signed and the data encrypted (enveloped data): a conventional session key is used to encrypt the data and that key is encrypted by the recipient's public key.
S/MIME is also used as Certificate management: it is a CA-centered system like SSL. An ordinary user is not aware of the CAs that he/she trusts, CA certificates come with the client software. Certificates are sent along with the signed messages in S/MIME.

35 TLS

35.1 Security for Network Centric Applications

The goals of information security are

- preserving integrity and/or confidentiality of data from sending point to the receiving point in a network
- providing for the receiver to be certain that the sender is actually the entity that the information was received from

These require:

- Authentication: when information is received from a source, the source is indeed as alleged in the information and the information was not altered along the way (integrity);
- Confidentiality: information is safe from being eaves-dropped on during its transit;
- Choosing best parameters: two endpoints of a communication link may possess different computational capabilities and may not have access to exactly the same set of security algorithms;
- Key management.

TLS provides Session Layer security in the OSI model of the internet protocols and plays a central role in the security and privacy needed for web commerce to work. Example: before your laptop uploads your credit card information to on-line shopping, it must make certain that the remote host is indeed what it claims to be.

A connection constitutes a peer-to-peer transient relationship between the two nodes: every connection is associated with a session. A session is an enduring association between a client and a server consisting of multiple connections and it is characterized by a set of security parameters that apply to all the connections in the session.

35.2 History of SSL/TLS

SSL (Secure Sockets Layer) was developed by Netscape in mid 1990s and originally shipped in Netscape Navigator 1.1. SSLv2 is now deprecated; SSLv3 still widely supported. It was developed as a way to secure communications between clients (browsers) and web servers.

TLS (Transport Layer Security) is the IETF-standardised version of SSL and the de facto standard for Internet security. It has the same SSL protocol design but different algorithms.

- TLS 1.0 = SSLv3 with minor tweaks, RFC 2246 (1999)
- TLS 1.1 = TLS 1.0 + tweaks, RFC 4346 (2006)
- TLS 1.2 = TLS 1.1 + more tweaks, RFC 5246 (2008)
- **TLS 1.3** = major changes + removed insecure features, RFC 8446 (2018)

It is now deployed in every web browser.

35.2.1 TLS in the browser

The lock icon is the SSL indicator. The intended goal is to provide users with identity of page origin and indicate to the user that page contents were not viewed or modified by a network attacker.

HTTPS (HTTP over TLS/SSL or HTTP Secure) provides:

- authentication of the website and associated web server with which one is communicating, to protect against man-in-the-middle attacks;
- bidirectional encryption of communications between a client and server, to protect against eavesdropping and tampering (or forging) with the contents of the communication (protection of privacy and integrity of the exchanged data).

35.3 TLS in a nutshell

As said in 34.6.2, it was designed originally for secure exchange of information between web servers and browsers, but more recently, SSL/TLS has become critically important to several other forms of information exchange, including exchange of information between:

- Routers
- routers and servers
- email exchange servers
- hosts and internet-accessible printers
- IoT devices and message brokers

When two endpoints involved in all the forms of information exchange have a need to authenticate each other and to create session keys for content encryption, they are likely to use SSL/TLS.

Certificates

Fundamental to the security that is established with SSL/TLS are the certificates issued by the Certificate Authorities (CA). Successful attempts at creating forged certificates undermine the security that can be achieved with SSL/TLS. SSL/TLS allows for either asymmetric (server-only) authentication or symmetric (server-client) authentication:

- **Server-only authentication**: client receives server's certificate, verifies the server's certificate and generates a secret key that it then encrypts with the server's public key. Client sends the encrypted secret key to the server, which decrypts it with its own private key and subsequently uses the client-generated secret key to encrypt the messages meant for the client.
- **Server-client authentication** (less common): in addition to the secret key, client also sends to the server its certificate, that the server uses to authenticate the client.

Verification of a certificate in TLS

As in 34.6.3.

35.3.1 Browser and certificate validation

Public keys of the root CAs are hard-coded into browsers. If a certificate cannot be validated by a browser in this manner (e.g., because the CA that has signed that certificate is not known to the browser) a warning popup will be generated by the browser. If user tells browser that he/she is willing to accept the certificate nonetheless, the authority that signed the certificate will be entered into the database of legitimate CAs maintained by the browser. Programming the keys of the root CAs into the browser code makes the root verification free of potential man-in-the-middle attacks. It is possible to check what root CAs are known to the browser by descending down the menu made available by the Preferences sub-menu under Editor.

35.3.2 A suite of protocols

TLS is not just a single protocol, but 4 protocols in 2 layers. The two most important are:

- **handshake**: authenticates clients and servers to each other (negotiate cipher suite, authenticate and establish keys used in the Record Protocol)
- **record**: transmits the data confidentially (provides confidentiality and integrity/authenticity of application layer data using keys from the Handshake Protocol)

The other two protocols (**Change Cipher Spec** and **Alert Protocols**) play relatively minor roles.

35.3.3 Connection state parameters

- Server Write MAC Secret: secret key is used in calculating the Message Authentication Code (MAC) value for the data sent by the server;
- Client Write MAC Secret: secret key is used in calculating the MAC value for the data sent by the client;
- Server Write Key: symmetric-key encryption key for data encrypted by the server and decrypted by the client;
- Client Write Key: symmetric-key encryption for data encrypted by the client and decrypted by the server;
- Initialization vectors: initialization vector (IV) for each key used by a block cipher operating in the Cipher Block Chaining mode is maintained;
- Sequence Numbers: each party maintains separate sequence numbers for the transmitted and received messages through each connection.

35.3.4 Session state parameters

- Session Identifier: arbitrary byte sequence chosen by the server to identify an active/resumable session state;
- Peer Certificate: X509.v3 certificate of the peer;
- Compression Method: algorithm used to compress the data prior to encryption;
- Cipher Spec: specifics of the bulk data encryption and hash algorithms used for MAC calculations;
- Master Secret: 48-byte secret shared between the client and the server;
- IsResumable: flag indicating whether the session is allowed to initiate new connections.

35.4 Record protocol

Provides confidentiality and (message) integrity. How?

On the **send** side:

- fragmenting the data into blocks;
- applying authentication and encryption primitives to each block;
- handing the block to TCP for transmission over the network.

On the **receive** side, these blocks are:

- decrypted;
- verified for message integrity;
- reassembled;
- delivered to the higher-level protocol.

35.4.1 Operations of the Record protocol

- **Fragmentation**: message (either from server to client, or from client to server) is fragmented into blocks whose length does not exceed 2^{14} (16384) bytes.
- **Compression** (optional): performs lossless compression and carries the stipulation that the size of the input block will not increase by more than 1024 bytes. Compression, in most cases, reduces the length of a block produced by the fragmentation step; but for very short blocks, the length may increase.
- **Adding MAC**: it computes the Message Authentication Code (like hashes) for the block that is appended to the compressed message block.
- **Encryption**: compressed message and MAC are encrypted using symmetric-key encryption.
- **Append Record Header**: a header is prepended to the encrypted block to declare content type, major version used for SSL/TLS, minor version used, and length of the (compressed, if the case) plaintext.

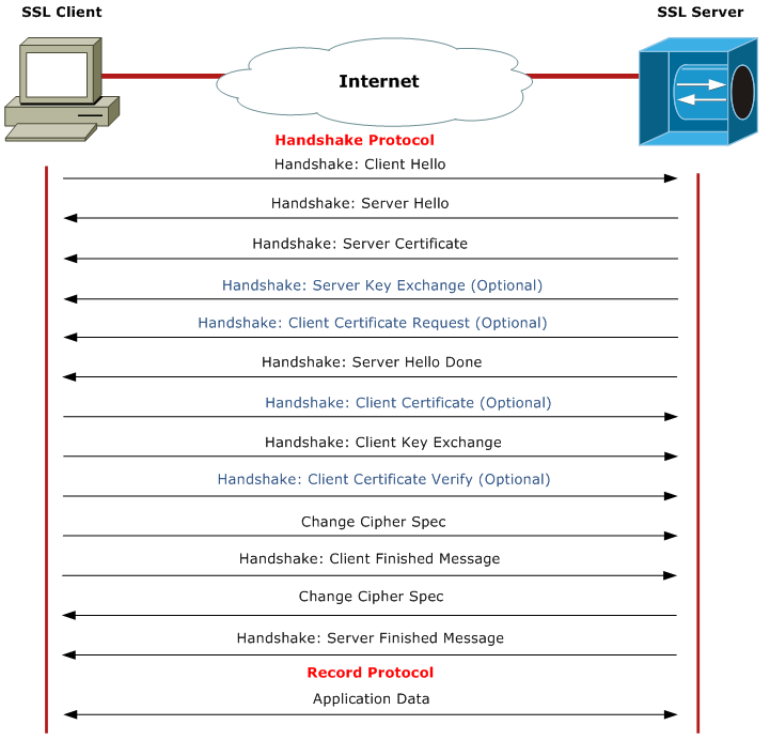
35.5 Handshake protocol

Provides necessary information for Record protocol to work (what algorithms to use for compression, authentication, and encryption). It is also responsible for:

- the server and the client to authenticate each other;
- to come up with the cryptographic keys to be used for the encryption and the authentication of each SSL/TLS record, i.e. each block produced by the Record protocol.

Organized in 4 phases:

1. Handshaking initiated by the client request
2. Handshaking initiated by the server sending certificate
3. Handshaking initiated by the client sending certificate (optional)
4. Handshaking completion



35.5.1 Phase 1

Handshaking, initiated by the client, establishes the security capabilities present at the two ends of a connection; client sends to the server a **client hello** message with the following parameters:

- Version (the highest SSL/TLS version understood by the client);
- Random (a 32-bit timestamp and a 28-byte random field that together serve as nonces during key exchange to prevent replay attacks);
- Session ID (a variable length session identifier);
- Cipher Suite (a list of cryptographic algorithms supported by the client, in decreasing order of preference);
- Compression Method (a list of compression methods the client supports).

Server responds with its **server hello** message that has a similar set of parameters and includes specific algorithms selected by the server from the client's lists for compression, authentication, and encryption. The Cipher Suite parameter, in particular, consists of two elements:

- One declares the key exchange method selected: choice is between RSA, three different types of Diffie-Hellman, etc;
- The other is called CipherSpec; it has a number of fields that indicate the authentication algorithm selected, the length of MAC, the encryption algorithm, etc.

35.5.2 Phase 2

Handshaking initiated by the server, that sends the **server certificate** to the client for the validation of its public key. From the user's perspective, when he wants the browser to upload credit-card information to an on-line shopping website, this is the most critical part of the handshake: the browser must make sure that the server at the other end is the real one and not someone else masquerading as the on-line shopping site. It establishes its trust in the server by validating the certificate downloaded from the on-line shopping server. Afterwards, server sends:

- A **server key exchange message**. May, e.g., consist of the global Diffie-Hellman values (a prime number and a primitive root of that number) and the server's DH public key;
- optionally a certificate request message if the server also wants to validate the client.

Ends when server sends client a **server hello done** message.

35.5.3 Phase 3

Handshaking initiated by the client, if server asked to, that sends certificate to the server for the validation of its public key. In most routine applications, the client will not send a certificate to the server. This is so because, while a client needs to authenticate the server if willing to share sensitive data (e.g., credit-card info), server has no real need to authenticate the client (e.g., it is sufficient for the server to check credit-card number).
The client sends to the server a mandatory **client key exchange** message, that could consist of a secret session key encrypted with the server's public key. This phase ends when the client sends to the server a **certificate verify** message to provide verification of its certificate (signed by a CA).

35.5.4 Phase 4

Handshaking completes the setting up of a secure connection between the client and the server. Client sends to the server a **change cipher spec** message, indicating that he's copying the pending CipherSpec into the current CipherSpec. Next, the client sends to the server the **finished** message. Finally, the server does the same vis-a-vis the client (**change cipher spec** + **finished**).

35.6 Notes on SSL/TLS state-of-the-art

35.6.1 Version and cipher suites

TLS versions used are SSL 3.0, TLS 1.0, TLS 1.1, TLS 1.2, TLS 1.3; some servers even still support SSL 2.0. There are more than 200 cipher suites: some very common, e.g.

`TLS_RSA_WITH_AES_128_CBC_SHA256`

some highly esoteric, e.g.

`TLS_KRB5_WITH_3DES_EDE_CBC_MD5`

35.6.2 Key establishment

TLS supports several key establishment mechanisms. The method used is negotiated during the Handshake Protocol itself. The client sends list of cipher suites it supports in ClientHello; server selects one and tells client in ServerHello. Cipher suites are encoded as 2-byte values. A common choice is RSA encryption:

- Server sends RSA public key and certificate chain after ServerHello;
- Client chooses pre_master_secret, encrypts using public RSA key of server, sends to server;
- RSA encryption is based on PKCS#1 v1.5 padding method.

35.6.3 Key establishment: replay attacks

There exist more options for key establishment including Diffie-Hellman and ECC. ClientHello offers many different cipher suites, choice of which to use is made by server; ClientHello and ServerHello also contain 32-byte nonces (arbitrary number that can be used just once in a cryptographic communication): 28-byte random values + 4-byte time encoding. Nonces are signed by the server in DH-based cipher suites and involved in key derivation. Nonce are important for security to prevent session replay attacks forcing reuse of session keys.

35.6.4 SSL/TLS version number

- Client: I support up to version x.
- Server: I will use version $y \leq x$.

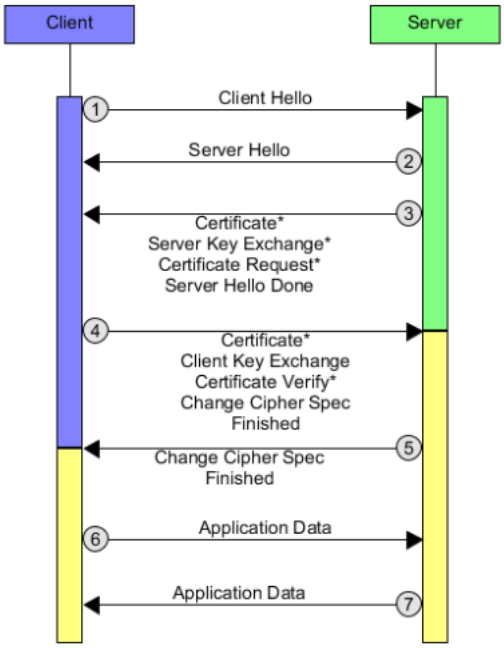
Legacy servers do not implement this correctly, simply failing if they do no support version x. Typical client behaviour is to try again with lower version in a fresh handshake (no memory of previous handshakes). The security consequence of this is that an active MITM can force client/server to roll back to the lowest SSL/TLS version they are both willing to use! POODLE attack exploits this to roll back to SSL3 and then perform Moeller attack on SSLv3 padding.

35.7 Other SSL/TLS protocols

- **Alert Protocol**:
  - Management of SSL/TLS connections and sessions, error messages;
  - Fatal errors and warnings;
  - Defined actions to ensure clean session termination by both client and server.
- **Change Cipher Spec Protocol**:
  - Technically not part of the Handshake Protocol;
  - Used to indicate that entity is changing to recently agreed cipher suite.

Both protocols run over Record Protocol (so are peers of Handshake Protocol).

35.8 TLS 1.2 Handshake protocol



Step 1

The **Client Hello** message contains the version of the protocol that the client wants to use and the list of supported cipher suite (set of algorithms that help secure a network connection that uses TLS/SSL).

Step 2

The **Server Hello** message contains the chosen protocol (supported by both parties), the chosen cipher suite (supported by both parties) and the **session ID**, a freshly-generated value that will identify the new session. A session is a series of interactions between two communication end points that occur during the span of a single connection. Typically, one end point requests a connection with another specified end point and if that end point replies agreeing to the connection, the end points take turns exchanging commands and data ("talking to each other").

Step 3

If the client requested an authenticated connection, the server must send a X.509 certificate. During the **Server Key Exchange**, a message sets the premaster secret that will be later used to generate the Master Secret. The **Certificate Request** message is sent by a non-anonymous server in case it requires the client to be authenticated.

Step 4

If the client has received a CertificateRequest message, it must send a X.509 certificate. During the **Client Key Exchange**, a message sets the premaster secret that will be later used to generate the Master Secret. The **Certificate Verify** is a message that provides explicit verification of the client certificate.

Step (4 and) 5

Change Cipher Spec is a message (consisting of a single byte of value 1) sent by both parties. Once received, the participant transitions to the agreed cipher suite. The **Finished** is generated by hashing the entire handshake and sent by both parties. It is used to signal the completion of the algorithm.

Step 6-7

From the reception of the Change Cipher Spec message onward, every message sent (both by server and client) will be encrypted using the shared key.

35.9 Security issues

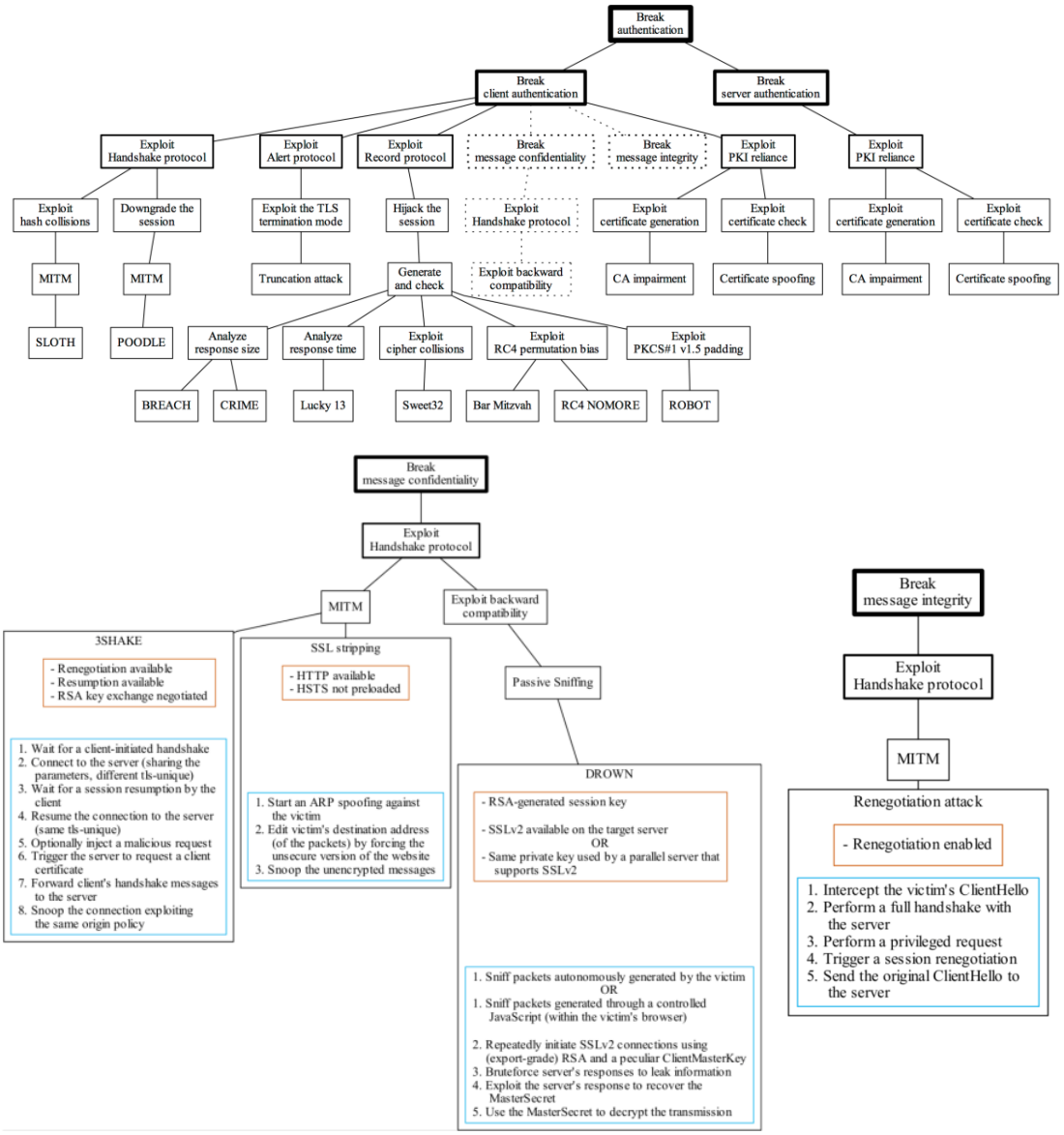
TLS might just seem like Handshake Protocol followed by Record Protocol. In reality, it is much more complex:

- The initial Handshake Protocol happens over Record Protocol with no keys;
- The Change Cipher Spec Protocol message, switch on new keys;
- After the completion of Handshake, marked by the exchange of Finished messages, the connection is running over a keyed Record Protocol;
- This is followed by arbitrary sequences of Session Resumption and Renegotiation runs.

Most of this activity is hidden from applications. This complexity has turned out to have negative security consequences. As a result TLS (up to v1.2) suffers security issues for a few reasons:

- **Backward compatibility**: the protocol still supports weak cipher suites and broken hash functions;
- **Logical flaws**: a set of logical loopholes can be used to "trick" both client and server.

The set of viable attacks can be structured as a tree:



35.10 Digression on cookies

HTTP is a stateless, application-level protocol for distributed, collaborative, hypermedia information systems. It is:

- **Connectionless**: a client (e.g., a browser) initiates an HTTP request and after a request is made, the client disconnects from the server and waits for a response. The server processes the request and re-establishes the connection with the client to send a response back;
- **Media independent**: any type of data can be sent by HTTP as long as both the client and the server know how to handle the data content;
- **Stateless**: the server and client are aware of each other only during a current request. Afterwards, both of them forget about each other. Due to this nature of the protocol, neither the client nor the browser can retain information between different requests across the web pages.

An HTTP **cookie** (also called web cookie, Internet cookie, browser cookie, or simply cookie) is a small piece of data sent from a website and stored on the user's computer by the user's web browser while the user is browsing. It is designed to be a reliable mechanism for websites to remember stateful information (such as items added in the shopping cart in an online store) or to record the user's browsing activity (including clicking particular buttons, logging in, or recording which pages were visited in the past). Authentication cookies are the most common method used by web servers to know whether the user is logged in or not, and which account they are logged in with; without such a mechanism, the site would not know whether to send a page containing sensitive information, or require the user to authenticate themselves by logging in. The security of an authentication cookie generally depends on the security of the issuing website and the user's web browser, and on whether the cookie data is encrypted. Security vulnerabilities may allow a cookie's data to be read by a hacker, used to gain access to user data, or used to gain access (with the user's credentials) to the website to which the cookie belongs (see cross-site scripting and cross-site request forgery for examples). There are **privacy** concerns: tracking cookies, and especially third-party tracking cookies, are commonly used as ways to compile long-term records of individuals' browsing histories. The European law requires all websites, targeting EU member states, to gain "informed consent" from users before storing non-essential cookies on their device.

Cookies can be exchanged in clear, or in the httpsOnly version sent by the server only over HTTPS: this allows **confidentiality** but not **integrity**, since a network attacker can rewrite cookies over http.

35.11 Example of attacks

Firesheep

Firesheep was an extension for the Firefox web browser that used a packet sniffer to intercept unencrypted session cookies from popular websites (e.g., Facebook and Twitter). The plugin eavesdropped on Wi-Fi communications, listening for session cookies, and when it detected one the tool used this cookie to obtain the identity belonging to that session. Collected identities (victims) were displayed in a side bar in Firefox and by clicking on a victim's name, the victim's session was taken over by the attacker. The extension was released October 2010, to demonstrate the security risks of **session hijacking** vulnerabilities on web sites that only encrypt the login process and not the cookie(s) created.

35.11.1 SSLStrip

An SSLStrip attack is an instance of **Men In The Middle** (MITM) attack. Instead of the victim connecting directly to a website, the victim would connect to the attacker, and the attacker would initiate the connection back to the website. In 2009 at Blackhat DC, it was shown the SSLStrip tool that would intercept HTTP traffic and whenever it spotted redirects or links to sites using HTTPS, it would transparently strip them away. Recall that there are two ways to follow https links:

- clicking on links with prefix https;
- via redirects (HTTP 302).

Attacker wants to read the traffic between two entities communicating via SSL/TLS. The idea underlying strip attack is to downgrade connection so that encryption is no more used. How is it possible to perform downgrading? Intercept web traffic and strip the 's' from all the links to https. The interceptor would make the encrypted connection back to the web server in HTTPS, and serve the traffic back to the site visitor unencrypted (logging any interesting passwords or credit card information in the process).

HSTS

To avoid this problem, a protocol called **HTTP Strict Transport Security** (**HSTS**) was created in 2012 and specified in RFC 6797. The protocol works by the server responding with a special header called Strict-Transport-Security which contains a response telling the client that whenever they reconnect to the site, they must use HTTPS. While this provided an improvement in preventing interception attacks, it was not perfect and there remain a number of shortcomings.

For example HSTS requires a previous connection to know to always connect securely to a particular site: when visitors first connect to the website, they will not have received the HSTS rule that tells them to always use HTTPS. Only on subsequent connections will the visitor's browser be aware of the HSTS rule that requires them to connect over HTTPS.

HSTS Preload Lists are one potential mitigations: they effectively work by hardcoding a list of websites that need to be connected to using HTTPS-only. Sites with HSTS enabled can be submitted to the Chrome HSTS Preload List at *hstspreload.org*, which is also used as the basis of the preload lists in other browsers. Inside the source code of Google Chrome, there is a file which contains a hardcoded file listing the HSTS properties for all domains in the Preload List.

35.11.2 Sweet32 - birthday attack

Known since long time, only recently (2016) the attack has become feasible. This attack allows to recover small portions of plaintext when encrypted with 64-bit block ciphers under some circumstances. 3DES roughly combines 3 times the symmetric block cipher DES (which is well known to be weak); by monitoring the victim's traffic and be able to execute Javascript in the victim's browser, it is possible to recover HTTP session cookies sent over TLS-encrypted channels in 1 or 2 days time. This is possible because block-ciphers (such as 3DES) can only encrypt a limited number of plaintext blocks before they are likely to produce a collision (on average after 32 GB of data).

Attack scenario

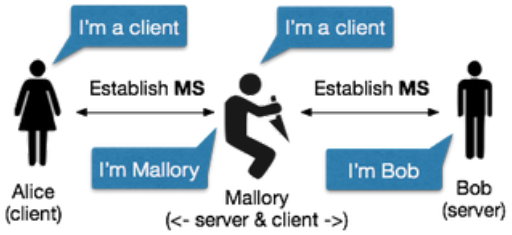
First, make an HTTP Basic Authentication:

- Enter login and passwords;
- Repeatedly send them over HTTP, with each request by the browser in a user transparent way.

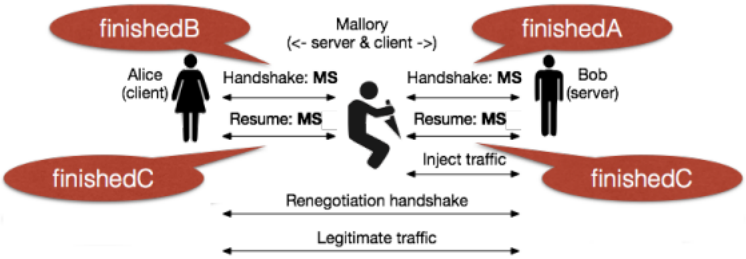
Install Javascript code in the victim's browser by inducing him/her to visit a malicious web site or install a malicious plug-in; the javascript code generates request containing fixed text + HTTP Basic Authentication request. After roughly 2^{32} requests (recall that a birthday attack may happen on average after $2^{(n/2)}$ for $n = 64$ bits in the case of 3DES), a collision is produced from which it is possible to derive the HTTP request.
To mitigate this, disable 3DES in TLS 1.2 (in TLS 1.3 it is no more supported).

35.11.3 3SHAKE

TLS Handshake protocol does a good job in establishing a common key but it does not guarantee the key to be unique. The idea underlying 3SHAKE is to exploit this lack of uniqueness by setting up two sessions using very similar artifacts (keys, session parameters, different certificates) and then to exploit the session resumption facilities to force the two similar sessions to share the same common keys.



Mallory mediates two handshakes: she can generate and force both connections to use the same **MasterSecret**. While the attacker has tricked the client into using a specific MS value, the handshake Finished messages (which the client will attach to the renegotiation handshake) will not be the same in both handshakes, because (among other things) the certificates sent on each connection were very different, hence the handshake hashes are not identical.
However, if the attacker triggers a session resumption (with a "session resumption handshake", to resume their session with new encryption keys), which uses no public-key cryptography or certificates at all (to make it faster), now the two sessions can be mistaken: both connections now see exactly the same (resumption) handshake messages, hence the hash of these handshakes will be identical, which means that their "Finished" message will be identical.



Within the resumed session, Mallory injects malicious traffic (e.g. "send money from Alice to Mallory"). By exploiting the renegotiation mechanism, Mallory merges the sessions so that the malicious request is executed into Alice's context. How to mitigate?
One proposed fix is to change the derivation of the Master Secret such that it includes the handshake hash. This should wipe out most of the attacks above. Another fix is to bind the "session resumption" handshake to the original handshake that led to it.

35.11.4 More on TLS vulnerabilities

TLSAssistant is an automated tool that combines state-of-the-art TLS analyzers with a report system that suggests appropriate mitigations and shows the full set of viable attacks. TLSAssistant is currently able to detect and provide mitigations for: 3SHAKE, Bar Mitzvah, BREACH, Client-Initiated Renegotiation DoS, CRIME, DROWN, HSTS not preloaded, HSTS not set, HTTPS not enforced, Lucky13, Missing Certificate Transparency, POODLE, RC4NOMORE, ROBOT, SLOTH, Sweet32, ...

36 SAML and co

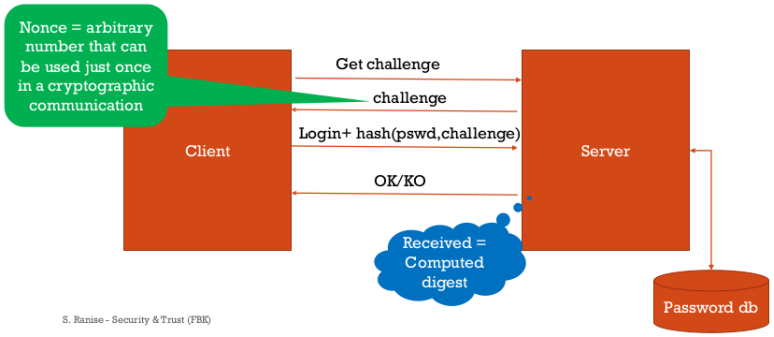
36.1 Remote authentication

In remote, the authentication over the network is more complex, there are problems of **eavesdropping** (unauthorized interception of a conversation, communication or digital transmission in real time) and **replay** (a valid data transmission is maliciously or fraudulently repeated or delayed). It is generally used a system of challenge-response:

1. user sends identity
2. host responds with random number r
3. user computes $f(r, h(P))$ and sends back
4. host compares value from user with own computed value, if match user authenticated

Remote authentication implies some security issues:

- **Client attacks**: attacker attempts to achieve user authentication without access to the remote host, masquerade as a legitimate user (e.g., guess the password or try all passwords). A countermeasure is the use of strong passwords and to limit the number of attempts;
- **Host attacks**: attacker attacks the host where passwords/passcodes are stored. A countermeasure is the use of hashing to protect the password databases;
- **Eavesdropping**: attacker attempts to learn passwords by observing the user, finding written passwords, keylogging. The countermeasures include diligence to keep passwords, multifactor authentication and the admin revoking compromised passwords;
- **Replay**: attacker repeats a previously captured user response. Countermeasure are the use of challenge-response and 1-time passcodes;



- **Trojan horse**: an application or physical device masquerades as an authentic application or device. Countermeasure is the authentication of the client within a trusted security environment;
- **Denial of service**: attacker attempts to disable a user authentication service (via flooding). Countermeasure is a multifactor authentication with a token.

36.2 Single Sign On (SSO)

Multiple systems typically require multiple sign-on dialogues. This is a password fatigue (multiple sets of credentials and presenting credentials multiple times) and it is an headache for administration and users. The more **security domains** (an application or collection of applications that all trust a common security token for authentication, authorization or session management), the more sign-ons required. A security token is issued to a user after the user has actively authenticated with a user ID and password to the security domain.

The idea behind SSO is to limit the issue of these tokens, so that the user has to log, and so insert his credentials, less times.

36.2.1 SSO properties

- Credentials never leave the authentication domain (i.e. identity provider + user agent);
- Service providers (affiliated domains) have to trust the authentication domain: credentials must be asserted correctly and protect from unauthorised use;
- Authentication transfer has to be protected: prevention of replay and interception/masquerade attacks.

36.2.2 SSO dependencies

SSO system relies on other infrastructure/procedures: the authentication system, the interface with web server and the identity management/registration. Applications often need more than just authentication information (attribute information).

36.2.3 Additional observations

- Most SSO systems are HTTP based (HTTP redirects);
- Most SSO systems rely on cookies, which are widely accepted and supported by browsers: users who disable cookies or change browser security settings may lose SSO capability;
- X.509 certificates provide alternative: require installation on users machine, need for revocation. Can be confusing for users;
- Session management: the SSO application maintains a session for the user, the target application usually maintains a session. Logging out of the target application may not log you out of the SSO application: Single Sign-On $\Rightarrow$ Single Sign-Out (application specific approach).

36.2.4 Federated identity and SSO

A federated identity is the means of linking a person's electronic identity and attributes, stored across multiple distinct identity management systems. Federated identity is related to single sign-on (SSO), in which a user's single authentication ticket, or token, is trusted across multiple IT systems or even organizations. SSO is a subset of federated identity management, as it relates only to authentication and is understood on the level of technical interoperability and it would not be possible without some sort of federation.

36.3 SAML

SAML (Security Assertion Markup Language) 1.0 was defined in 2002, SAML 2.0 in 2005. It is a common language and flow between systems that want to provide an SSO experience to users and answer to the lack of standards and interoperable solutions for exchanging authentication and authorisation information across security domains. It is XML-based, an open standard and a product of OASIS; its main goals are Cross-Domain Single Sign-On (SSO) and Identity federation.

SAML simplifies federated authentication and authorization processes for users, Identity providers, and service providers. It provides a solution to allow an identity provider and service providers to exist separately from each other, which centralizes user management and provides access to SaaS solutions. SAML implements a secure method of passing user authentications and authorizations between the identity provider and service providers: when a user logs into a SAML enabled application, the service provider requests authorization from the appropriate identity provider, that authenticates the user's credentials and then returns the authorization for the user to the service provider, and the user is now able to use the application.

36.3.1 SAML federation

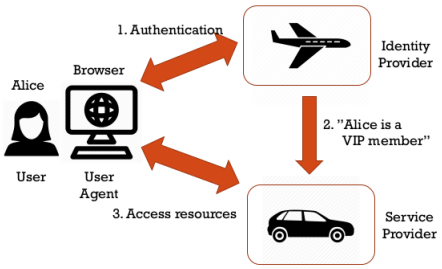
SAML distinguish two main resources:

- Identity Provider (IdP): authenticates the user and provides authorisation information;
- Service Provider (SP): a server that hosts protected resources, relies on information provided by the IdP and has local access policies to regulate access to protected resources.

A federation is a group of resources sharing a common policy and managed as a single entity. The authentication is always between an IdP and a SP. The federation establish only the initial trust among the resources.

Possible scenario

Consider Alice visits an airline website for making her trip. For booking her flight, she provides her credentials to the airline website. After booking, she found a link to a car rental (say from the airline website) and she visits its website. Alice can rent a car without signing in again because the car rental site trusts the airline site when transmitting authentication assertions such as 2.



36.3.2 SAML authentication

1. A user want to access an SP
2. The user is redirect to a Discovery Service, that can be an external service or embedded in the SP, and allows the user to choose the IdP
3. The user goes back to the SP with the ID of the own IdP
4. The user is redirect to the IdP
5. Authentication is performed
6. The user goes back to the SP with the authentication

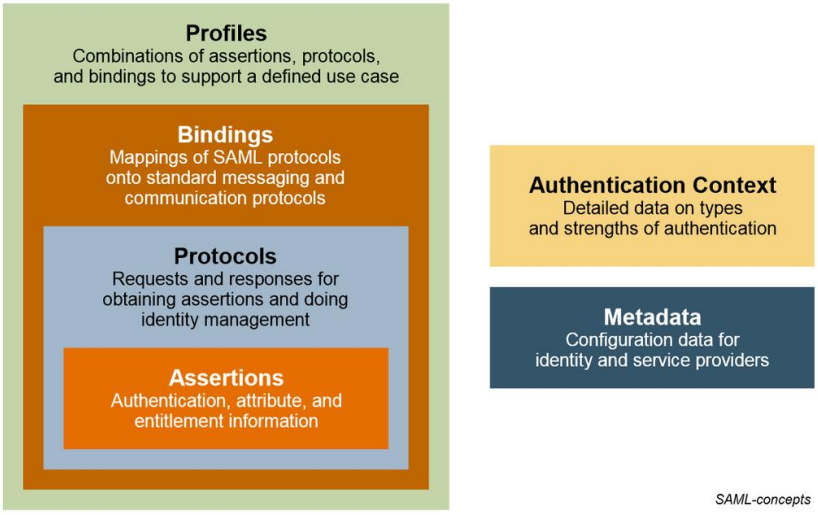
All the steps above are associated with a SAML assertion (in XML format). SAML Authentication was mainly designed for Web Services. It is possible to implement other type of resources, either web based or native application. Resources can support multiple SAML profiles: the profile identifies the exchange protocol and the message format. The most used profile for web application is redirect: the browser show a page with a Javascript performing the redirect.

IdP keeps tracks of user authentications. Multiple requests from different services require to authenticate only once, although services can require different information. Attribute release has to be confirmed and authorised by the user: this is not implemented in all the IdP software. Discovery service could store the user selection: if this is the case, users could have problem to change the IdP in the future.

36.3.3 SSO basic provisions and flows

- Cross-Domain SSO: a standard vendor-independent protocol for transferring information across domains. It does not rely on cookies;
- Federated identity: sharing information about user identities across organisations;
- Two types of flows: **push** and **pull**
  - **IdP-initiated** (push): IdP authenticates first (airline and renting car example follows the IdP-initiated flow);
  - **SP-initiated** (pull): an SP requests the IdP to authenticate the Subject.

36.3.4 SAML overview



Assertions

Assertion is the unit of information in SAML. Assert characteristics and attributes of a Subject:

- 'Alice' is a 'VIP member';
- Her email is 'alice@example.com';
- She is a member of the 'Engineering' group.

There are three types of assertions:

- Authentication statement: issued by a party that authenticates users. It describes who issued the assertion, the authenticated Subject, the validity period and other authentication-related information;
- Attribute statement: it defines specific details about the Subject;
- Authorisation decision statement: it defines something the Subject is entitled to do.

Protocols

- Flow of assertion query and request (for obtaining SAML assertions);
- Authentication request;
- Artifact resolution (a mechanism by which protocol messages may be passed by references);
- Single logout;
- and much more...

Bindings

SAML requestors and responders communicate by exchanging messages; the mechanism to transport these messages is called a SAML binding. Types of this are: SAML URI, SAML SOAP, ..., HTTP redirect, HTTP POST, HTTP artifact. **HTTP redirect** enables SAML protocol messages to be transmitted within URL parameters: it enables SAML requestors and responders to communicate by using an HTTP user agent as an intermediary. The intermediary might be necessary if the communicating entities do not have a direct path of communication. The intermediary might also be necessary if the responder requires interaction with a user agent, such as an authentication agent. HTTP redirect is sometimes called browser redirect in single sign-on operations. This profile is selected by **default**.

Profiles

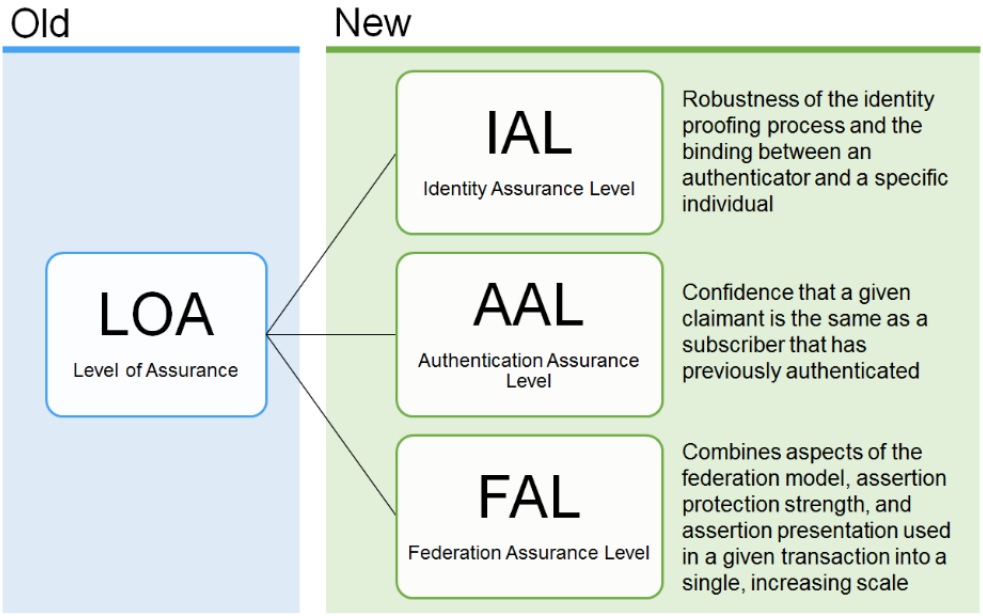
SAML 2.0 profiles combine protocols, assertions, and bindings to create a federation and enable federated single sign-on. Types of profiles: Web browser SSO, Single Logout, Artifact resolution, ...

- **Web browser single sign-on** profile provides options regarding the initiation of the message flow and the transport of the messages. The message flow can be initiated from the identity provider or the service provider and the following bindings can be used: HTTP redirect, HTTP POST, HTTP artifact. A pair of partners (IdP and SP) in a federation does not need to use the same binding.
- **Single Logout** profile is used to terminate all the login sessions currently active for a specified user within the federation: a user who achieves single sign-on to a federation establishes sessions with more than one participant in the federation. The sessions are managed by a session authority, which in many cases is an identity provider. When the user wants to end sessions with all session participants, the session authority can use the single logout profile to globally terminate all active sessions.

Authentication context

It indicates how a user authenticated at an Identity Provider. The Identity Provider includes the authentication context in an assertion at the request of a Service Provider or based on configuration at the Identity Provider. A Service Provider can require information about the authentication process to establish a level of confidence in the assertion before granting access to resources. The motivation is due to the fact that existing SAML federation deployments have adopted a "Levels Of Assurance" (or LOA) model for categorizing the wide variety of authentication methods into a small number of levels, typically based on some notion of the strength of the authentication. Service Providers then decide which level of assurance is required to access specific protected resources, based on some assessment of "value" or "risk".

NIST LOA Levels



Identity Assurance Level

Refers to the robustness of the identity proofing process and the binding between the authenticator (something the claimant possesses and controls, that is used to authenticate the claimant's identity) and a specific individual.

IAL	Description
1	Self-asserted attributes - 0 to n attributes
2	Remotely identity proofed
3	In-person identity proofed

Authentication Assurance Level

Refers to the confidence that a given claimant is the same as the subscriber who has previously authenticated.

AAL	Description
1	Single-factor authentication
2	Two-factor authentication
3	Two-factor authentication with hardware authentication

Federation Assurance Level

Combines aspects of the federation model, assertion protection strength, and assertion presentation used in a given transaction.

FAL	Presentation requirement
1	Bearer assertion, signed by IdP
2	Bearer assertion, signed by IdP and encrypted to RP
3	Holder of key assertion, signed by IdP and encrypted to RP

Metadata

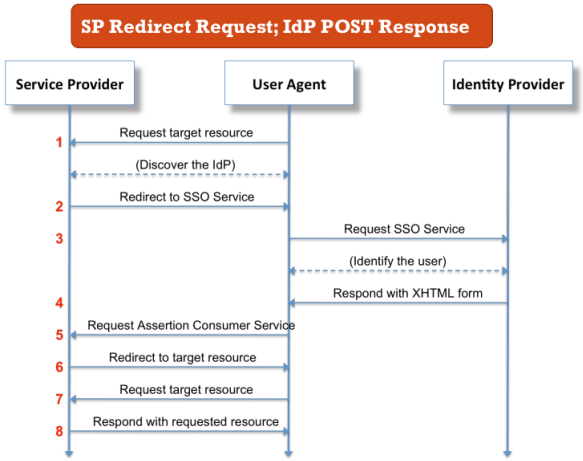
A SAML metadata document describes a SAML deployment such as a SAML identity provider or a SAML service provider. Deployments share metadata to establish a baseline of trust and interoperability. The minimum set of metadata to be shared is:

- Entity ID
- Cryptographic Keys
- Protocol Endpoints (bindings and URLs)

Every SAML system entity has an entity ID, a globally unique identifier used in software configurations, relying-party databases, and client-side cookies. On the wire, every SAML protocol message contains the entity ID of the issuer. For authentication purposes, a SAML message may be digitally signed by the issuer. To verify the signature on the message, the message receiver uses a public key known to belong to the issuer. Similarly, to encrypt a message, a public encryption key belonging to the ultimate receiver must be known to the issuer. In both situations (signing and encryption) trusted public keys must be shared in advance. Once the message is signed and encrypted, the issuer sends the message to a trusted protocol endpoint, the location of which must be known in advance. Upon receipt, the message receiver decrypts the message (using its own private decryption key) and verifies the signature (using a trusted public key in metadata) before mapping the entity ID in the message to a trusted partner. This scenario requires each party to know the other in advance. To establish a baseline of trust, parties share metadata with each other. Initially, this may be as simple as sharing information via email; over time, as the number of SAML partners grows, the natural tendency is to automate the metadata sharing process.

An implementation that supports SAML, Web Browser SSO, requires a schema-valid SAML metadata file for each SAML partner.

Web Browser SSO



It is one of the most common scenarios: the service provider sends a SAML Request to the IdP SSO Service using the HTTP-Redirect Binding, that is suitable for short messages (since the length of the URL is limited). The identity provider returns the SAML Response to the SP Assertion Consumer Service using the HTTP-POST Binding. Discovering the IdP is possible because metadata have been preliminarily shred.

1. **Request the target resource at the SP.** The principal (via an HTTP user agent) requests a target resource at the service provider: `https://sp.example.com/myresource`. The service provider performs a security check on behalf of the target resource. If a valid security context at the service provider already exists, skip steps 2–7. The service provider may use any kind of mechanism to discover the identity provider that will be used, e.g., ask the user, use a preconfigured IdP, etc.

2. **Redirect to IdP SSO Service.** The service provider generates an appropriate SAMLRequest (and RelayState, if any), then redirects the browser to the IdP SSO Service using a standard HTTP 302 redirect:

```
https://idp.example.org/SAML2/SSO/Redirect?
        SAMLRequest=request&RelayState=token
```

The RelayState token is an opaque reference to state information maintained at the service provider. The value of the SAMLRequest parameter is a deflated, base64-encoded and URL-encoded value of a <samlp:AuthnRequest> element:

```
<samlp:AuthnRequest xmlns:samlp=
    "urn:oasis:names:tc:SAML:2.0:protocol"
xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
    ID="identifier_1" Version="2.0" IssueInstant=
        "2004-12-05T09:21:59Z"
    AssertionConsumerServiceIndex="0">
<saml:Issuer>https://sp.example.com/SAML2</saml:Issuer>
<samlp:NameIDPolicy AllowCreate="true"
Format="urn:oasis:names:tc:SAML:2.0:nameid-format:transient"/>
</samlp:AuthnRequest>
```

The SAMLRequest may be signed using the SP signing key.

3. **Request the SSO Service at the IdP.** The user agent issues a GET request to the SSO service at the identity provider:

```
GET /SAML2/SSO/Redirect?SAMLRequest=request
    &RelayState=token HTTP/1.1
Host: idp.example.org
```

where the values of the SAMLRequest and RelayState parameters are the same as those provided in the redirect. The SSO Service at the identity provider processes (by URL-decoding, base64-decoding and inflating the request, in that order) the <samlp:AuthnRequest> element and performs a security check. If the user does not have a valid security context, the identity provider identifies the user with any mechanism.

4. **Respond with an XHTML form.** The SSO Service validates the request and responds with a document containing an XHTML form:

```
<form method="post" action=
    "https://sp.example.com/SAML2/SSO/POST" ...>
<input type="hidden" name="SAMLResponse" value="response" />
<input type="hidden" name="RelayState" value="token"/>
...
<input type="submit" value="Submit" /> </form>
```

The value of the RelayState parameter has been preserved from step 3.

5. **Request the Assertion Consumer Service at the SP.** The user agent issues a POST request to the Assertion Consumer Service at the service provider:

```
POST /SAML2/SSO/POST HTTP/1.1 Host: sp.example.com
    Content-Type: application/x-www-form-
urlencoded Content-Length: nnn
    SAMLResponse=response&RelayState=token
```

where the values of the SAMLResponse and RelayState parameters are taken from the XHTML form at step 4.

6. **Redirect to the target resource.** The assertion consumer service processes the response, creates a security context at the service provider and redirects the user agent to the target resource.

7. **Request the target resource at the SP again.** The user agent requests the target resource at the service provider (again): `https://sp.example.com/myresource`.

8. **Respond with requested resource.** Since a security context exists, the service provider returns the resource to the user agent.

36.4 Possible to use cookies instead?

36.4.1 Cookies

An HTTP cookie is a small file left in a Web browser by a Web server, that carries a name-value pair (its useful payload) and a set of management attributes. Cookies were introduced by Netscape as a state management mechanism to the otherwise stateless HTTP protocol in 1994. The current implemented standard for HTTP cookies is RFC 2109; it defines the cookie format and the rules for proper handling of cookies by Web browsers, servers, and proxies.

attribute	description
Comment	Short description of the intended use of the cookie
Domain	DNS domain or IP address for which the cookie is valid
HttpOnly	If present, the cookie cannot be accessed by a client-side script (e.g., written in JavaScript). Although non-standard, this attribute is supported by most Web browsers
Max-Age	Maximum period after which the cookie must be discarded
Path	Subset of URLs on qualifying hosts for which the cookie is valid
Port	List of TCP ports on qualifying hosts for which the cookie is valid
Secure	If present, the cookie may be transported only over a secure (e.g., SSL-protected) channel

Cookies may be subdivided into:

- session cookies: erased when the browser is closed;
- persistent cookies: preserved across multiple browser sessions.

Persistent cookies are used as an inexpensive way to store users' preferences for a Web site; the advantage is that the Web site operator does not need to maintain an account for the user; all pertinent information is stored in the browser, and made available to the site when the user visits it. Browsers use the Same Origin Policy to determine whether to send a cookie to a Web site: a HTTP request sent to a host will contain those and only those cookies whose Domain attribute identifies the host itself or the DNS domain to which the host belongs. One of the requirements of the **same origin policy** is that cookies be shared only between Web sites within the same administrative domain. Subsequent standards did not change this basic assumption. SSO uses browser cookies to maintain state so that re-authentication is not required, but browser cookies are not transferred across domains. Using assertions, SAML offers SSO across domains. In conclusion, cookies cannot replace SAML assertions!

36.5 An attack to SAML

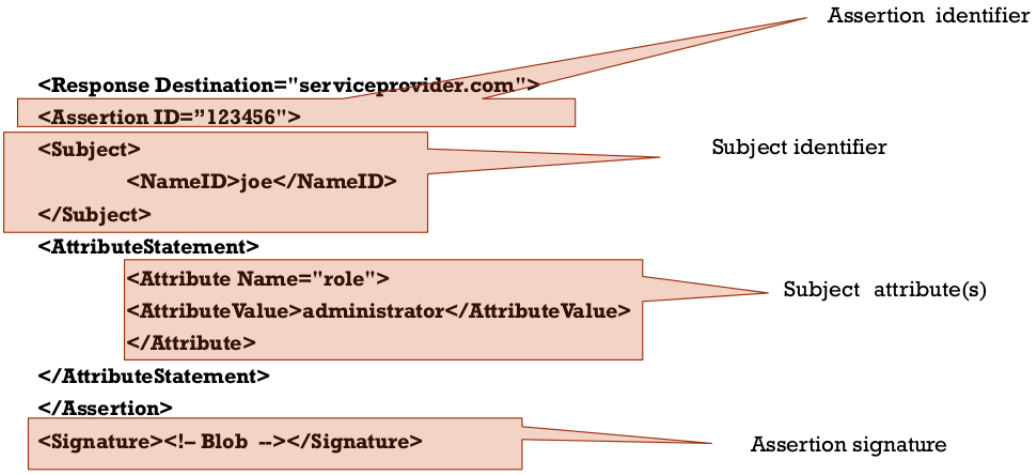
Recap

- **SSO experience**: authenticate once, access to multiple applications. This implies an improvement of user experience on N-passwords for N-applications (reduced password fatigue);
- **Three entities involved in SSO**
  - Identity Provider (IdP): the service at which users authenticate

  - User Agent (UA): web browser used for message passing (redirections)
  - Service Provider (SP): the service users want to get access to

- **SSO flow**: once authenticated, IdP passes a message through browser, which takes the message and passes it to the SP. The SP validates the message and then authenticates you to the service.

A SAML assertion



Signature element

```
<Assertion ID="123456">[...]</Assertion>
<Signature Id="signature_id">
<SignedInfo>
    <CanonicalizationMethod Alg="xml-c14n11"/>
    <SignatureMethod Algorithm="rsa-sha256"/>
    <Reference URI="123456">
    <Transforms>
        <Transform Algorithm="xmldsig#base64"/>
    </Transforms>
    <DigestMethod Algorithm="sha1"/>
    <DigestValue>eW8=</DigestValue>
</Reference></SignedInfo></Signature>[...]
```

36.5.1 XML canonicalization

XML canonicalizes data prior to signature operations: logically equivalent documents have the same signature. The creation of XML Signatures is substantially more complex than the creation of an ordinary digital signature because a given XML Document may have more than one "legal" representation. Example: whitespace inside an XML Element is not syntactically significant, so that <Elem > is syntactically identical to <Elem>. Since the digital signature ensures data integrity, a single-byte difference would cause the signature to vary. There are also interoperability issues: if an XML document is transferred from computer to computer, the line terminator may vary: a program that digests and validates an XML document may later render the XML document in a different way, e.g., using relative (vs. absolute) URLs. To avoid these problems and guarantee that logically-identical XML documents give identical digital signatures, an XML canonicalization (abbreviated C14n) is employed when signing XML documents: for signing the SignedInfo, a canonicalization is mandatory. XML canonicalization algorithms guarantee that semantically-identical documents produce exactly identical representations.

36.5.2 SAML library

SAML defines a "language" to convey data. What to do with the data is up to implementers; APIs often provide non-canonical doc to users after verification.

Text extraction

```
saml_response = init_saml_auth(http_request)
saml_response.process_response()
errors = saml_response.get_errors()
if len(errors) == 0:
    user_id = saml_response.get_nameid()
    authenticate(user_id)
else:
    # user is not authenticated
```

Consider the string `"<NameID>ranise@fbk.eu</NameId>"` is in `saml_response`, what is the content of `user_id` after the execution of `get_nameid`? It is the string *"ranise@fbk.eu"*.

Now, consider the string `"<NameID>ranise<!---->@fbk.eu</NameId>"` is in `saml_response`, what is the content of `user_id` after the execution of `get_nameid`? With some APIs, it is again the string *"ranise@fbk.eu"*!

36.5.3 Recap

SAML documents:

- Contain information about the authenticating user, within the inner text;
- Are often passed through a user's browser from the IdP to the SP;
- Are signed to prevent tampering;
- Are canonicalized, in most cases excluding comments, prior to signing;
- Because of canonicalization, the addition of comments would not affect a document signature.

SAML APIs:

- Have unintuitive and/or inconsistent behavior when extracting inner text;
- Provide the non-canonical document representation for post-signature processing.

36.5.4 An attack

```
<NameID>
    admin@victim.com.evil.com
</NameID>
```

This can be edited to:

```
<NameID>
    admin@victim.com<!--inserted-->.evil.com
</NameID>
```

This does not affect signature validity, becuase it is truncated by the SP. Idea: use own credential to pivot to other user accounts.

To detect the vulnerabilities use tests such as:

```
doc = "[...]<NameID>us<!---->er</NameID>[...]"
doc.verifySignature()
assert doc.nameID == "user"
```

If assertion fails, then NameID is likely to be truncated.

Exploitation of SP and IdP

The attacker has an account on Identity Provider so he uses foothold to gain access to Service Providers and uses the XML comment bug to pivot into Service Providers for different users.
Does the SAML Response convey role information? SAML defines the AttributeStatement element which can contain fairly generic attribute values and can be used to define role information:

```
<AttributeStatement>
<Attribute Name="Groups">
    <AttributeValue>Administrators (HR)</AttributeValue>
    <AttributeValue>Employees</AttributeValue>
</Attribute>
</AttributeStatement>
```

36.6 SPID

Sistema Pubblico di Identità Digitale (SPID) is the Italian Public System of Digital Identity based on SAML 2.0. It was developed to control the proliferation of credentials distributed by different Public Administrations, that caused difficulty in updating passwords on certain web sites and difficulty in remembering all passwords. SPID is a single set of credentials to access any Public Administration service: there is a obligation for all public administrations to allow its usage.

36.6.1 Mandatory attributes

- Natural persons: surname and name, sex, date and place of birth, tax code, details of a valid identity document, a mobile phone number and an e-mail address;
- Legal persons: company name, tax code or VAT number, registered office, Chamber of Commerce perusal attesting the status of legal representative of the person requesting the identity on behalf of the company (alternatively notary deed of legal power of attorney) details of the identity card used by the legal representative, a mobile phone number and an e-mail address.

36.6.2 AgID (Agency for Digital Italy)

AGID controls **accreditation** and **supervisory activities** to activate SPID:

- Management of accreditation applications
- Conclusion of agreements with Identity Providers
- Updating the SPID register
- Evaluation of the technical/procedural activities
- Handle malfunction and security incidents
- Publish statistics

36.6.3 Identity providers

The IdPs are required to use reliable systems that guarantee the technical and cryptographic security of procedures, in accordance with international standards and AgID guidelines. They must adopt appropriate measures to ensure confidentiality, integrity and security in generating access credentials, continuously monitoring the use of credentials to prevent or detect attempts of violation or improper use while respecting the criteria of necessity and proportionality to protect the privacy of users. Every two years, they are subject to verification of compliance with current provisions. They must disclose to AgID:

- Intention to make changes
- Any violation/intrusion to user personal data
- Data on provided service (statistics and service level agreements)

36.6.4 Service providers

Service Providers have retention of information on the activities performed by users for 2 years, they have to communicate anomalies to AgID and IdPs, respect user's privacy and verificate the attributes.

36.6.5 Identity proofing

Identification is obtained through submission for visual inspection of a valid identity document by the applicant, who will sign the SPID application form. Digital identification is obtained through legally valid digital identity documents, which need for visual inspection of the applicant, including national electronic identity card, national service card and cards compliant with this, like the national electronic health card. Digital identification otherwise can be made through another SPID identity, issued by the same IDP, of an equal or higher security level than that being applied for. Acquisition of the SPID application form subscribed with a qualified electronic signature + a valid identity document and the national health card by the applicant (all verified on authoritative source). The authoritative source is the SCIPAFI, the public system for the prevention of fraud in the consumer credit sector with specific reference to identity theft, implemented by the Ministry of the Economy and Finance.

36.6.6 Authentication process

- Choose and IdP
- At the "low" level (**SPID level 1**), the authentication method consists of something that the user knows: username and password. Passwords must comply with precise robustness characteristics (minimum length, special characters, not easily referable to the person) and must be changed every 180 days; they are blocked after 5 incorrect attempts. Communications between the user and the authentication system are encrypted with the latest stable version of the HTTPS Protocol. The SAML assertions, which contain a time reference, are signed in order to guarantee authenticity and integrity. The URLs are written in the metadata and are not exchanged in authentication messages to prevent phishing attacks. Credentials are stored with encryption methods (SHA-512 hash salted function). Server access is only possible from the internal network with administrator credentials.
- At the "substantial" SPID level (**SPID level 2**) a one time password (OTP) is added to username & password. User can choose from different methods:
  - text message;
  - iOS, Android, Windows Phone app;
  - Physical Event-Based OTP Token or Physical Time-Based OTP Token

  One time password, of course, has a limited time value and can be used for one access only, with a maximum number of attempts. On expiry of the OTP or at the last attempt, the SAML response is generated and the authentication process is restarted. Even analysing a long sequence of valid OTPs, it is not possible to deduce the sequence. The OTP requires possession of a device associated with the user: the IdP therefore verifies actual possession of such device by the user. The OTP generated can be used in a single user session only.
- At the "high" level (**SPID level 3**), the authentication system is based on a double authentication factor plus a digital certificate, asymmetric encryption and the use of a device with the characteristics envisaged by Annex II of the eIDAS Regulation. Authentication therefore consists of a username, a password (SYK), and an OTP (SYH), plus the use of a device such as a smart card or HSM.

36.6.7 Prior identities

Prior identities in the possession of public administrations or private entities acquired over time in order to identify customer/ employees for service delivery, may be reused in order to obtain SPID identity, subject to the authorisation of AgID. Example with banks: banks which intend to make a Digital Identity available to customers accessing the Home Banking platform of the Bank via access credentials and who are in possession of the second authentication factor (e.g.: Token), may enter into a contract, delegating an IdP accredited with AgID. The IdP at this point, as envisaged by national legislation, will submit a specific application to AgID signed with qualified electronic signature (FEQ), attaching the documentation relating to the previous system for which evaluation is requested.

36.6.8 Technical rules for IDPS

- Identity provider manages users and the authentication procedure;
- Service provider, after having requested authentication of the user to the Identity Provider, manages the authorization, based on the attributes returned by the IdP, delivering the requested service;
- Attribute Authority provides certified attributes based on the authenticated user;
- The operating modes of the Identity provider, must be those envisaged by SAML v2 for the "Web Browser SSO" profile: the authentication mechanism is triggered by the request sent by the user (through its User Agent) to a Service Provider, which in turn contacts the appropriate Identity Provider in pull mode with a request based on the <AuthnRequest> construct, using the binding HTTP Redirect or the binding HTTP POST. The Identity Provider will only respond to the service provider via the HTTP POST binding with a message based on the <Response> construct.

36.6.9 Attribute authority

An Attribute Authority must be able to certify a given set of attributes relating to a holder of a digital identity. Upon receipt of a request for one or more attributes, it must be able to:

- receive and interpret the request for attribute received by a Service Provider;
- process the request;
- construct the response relating to the request received and forward it to the Service Provider.

36.6.10 SPID register

Repository of all the information related to the entities adhering to the SPID and represents the evidence of the so-called circle of trust established therein. The relationship of trust on which the federation established in SPID is based is achieved through the intermediation of the Agency, third party guarantor, through the process of accreditation of digital identity providers, the attribute authorities and service providers. Adhesion to SPID constitutes the establishment of a relationship of trust with all existing members accredited by the Agency, based on the sharing of the standard security levels of assurance declared and guaranteed by SPID. Adhesion to the trust agreement among member entities is demonstrated by the presence of such entities in the SPID Register managed by the Agency. The **federation registry** contains the list of entities that have passed the accreditation process and are therefore part of the SPID federation. For each entity the registry contains an entry called AuthorityInfo consisting of:

- SAML identifier of the entity
- name of the subject to which the federation entity refers
- type of entity (Identity Provider, Attribute Authority, Service Provider)

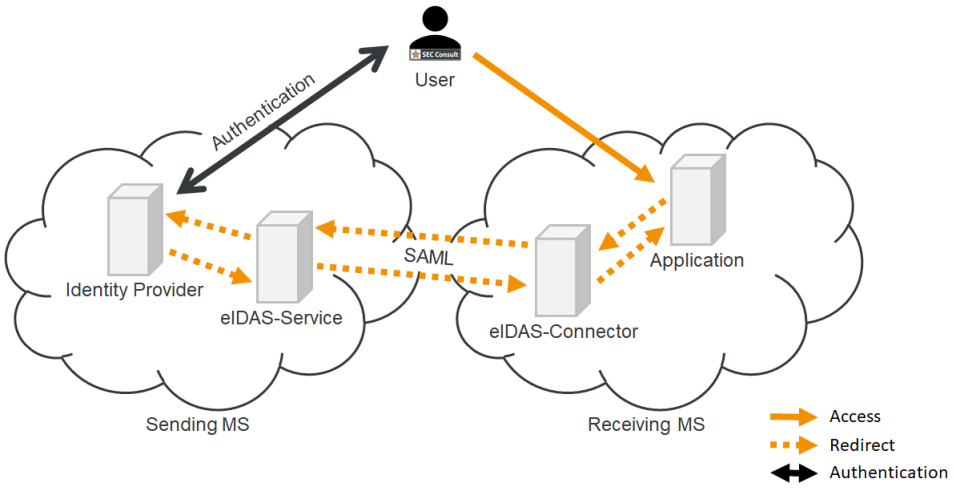
- URL of the metadata provider service
- List of qualified attributes which can be certified by an Attribute Authority

The federation registry is populated by AgID following stipulation of the agreements and updated by said Agency during the activities related to management of the agreements and the supervision of the parties of the SPID circuit.

It turns out that few Service Providers implement the Single Logout procedure in the right way: usually, a SP closes only its session but not the SSO session. This may have some security consequences. SSO at SPID level 2 (substantial) is a bit delicate as stated in the legal documents, it seems to be impossible to enter only once the credentials (username, password, and OTP) and have access to any resource or perform any operation. Entering credentials should be done for every access (resource or operation): this has usability consequences.

36.6.11 EIDAS (Electronic Identification, Authentication and trust Services)

SPID is integrated into the eIDAS Interoperability Framework. The Italian eIDAS-Node, hosted by AgID, is currently operational and it is based on the eIDAS reference. In order to provide a cross-border authentication mechanism, the Italian eIDAS-Node is based on "Proxy to Proxy“ architecture. The FICEP (First Italian Crossborder eIDAS Proxy) project enables Italian citizens to access online services from other European countries using the credentials obtained in the SPID public digital identity system. At the same time, European citizens with national digital identities recognised in eIDAS will be able to access the services of Italian Public Administrations.



If an Italian citizen wants to authenticate against a German online service, first the German eIDAS-Node (eIDAS-Connector) is directed by the web application to initiate the authentication process; it sends a request to the Italian eIDAS-Node (eIDAS-Service). The Italian eIDAS-Node forwards the user to a system that is equipped to authenticate the Italian citizen using the national eID scheme. After authentication, the German eIDAS-Connector receives the citizen's information which it forwards to the web application. eIDAS relies on SAML for communication between the eIDAS-Connector and eIDAS-Service (called eIDAS-Nodes).

36.7 CIE 3.0

Carta Identità Elettronica is an identification document containing personal data: picture and fingerprints; it enables authentication to on-line services containing keys and X.509 certificates. CIE can be used as a 2nd factor authentication (example in badging), generating OTP by challenge-response mechanism, or as a single authenticator, using the smartphone's browser.

37 OAuth and OpenID Connect

- OAuth 2.0 is for service authorization (delegation): authorize one website (consumer) to access data from another website (provider) – in a restricted manner. It was designed initially for web services. Example: Allow the NYTimes to tweet a message from Twitter account.
- OpenID Connect is for remote authentication (with Single-Sign-On experience): gives the possibility to use one login and password for accessing multiple services. It is built on top of OAuth 2.0.

37.1 OAuth 2.0

Its goal is to allow service authorization. The user wants an app to access (a selection of) his/her data stored in a given service S (e.g., allow printing service to access pictures stored in some server) without revealing passwords of service S, plus restrict data and operations available to the app (e.g., only retrieving a sub-set of the pictures stored). The user must be capable to revoke the app's access to the data.

OAuth is based on:

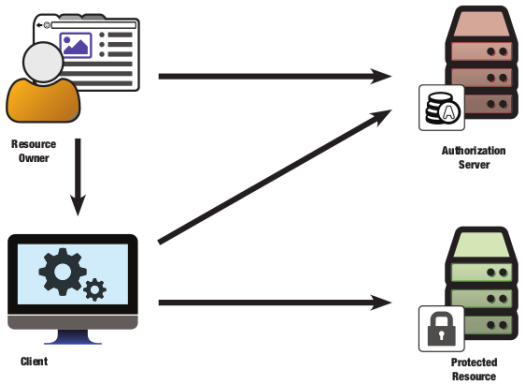
- getting a token from the service provider and presenting it each time an application accesses an API at the service;
- URL redirection: technique for making a web page available under more than one URL address;
- JSON data encapsulation: JavaScript Object Notation is an open-standard file format that uses human-readable text to transmit data objects consisting of attribute–value pairs and array data types (or any other serializable value).

The service provider (e.g., service storing pictures):

- Gets data about application (name, creator, URL);
- Assigns the application (consumer) an ID and a secret;
- Presents list of authorization URLs and scopes (= access types).

The entities involved are:

- **Resource owner**: usually is a person and can access or delegate access to certain resources;
- **Protected resource**: service provider protecting resources for their owner, shares them on owner's request;
- **Client** (app): wish to access protected resources and acts on the owner's behalf;
- **Authorization server**: generates tokens for the client (**token endpoint**), authenticate resource owners and clients, manages authorizations.



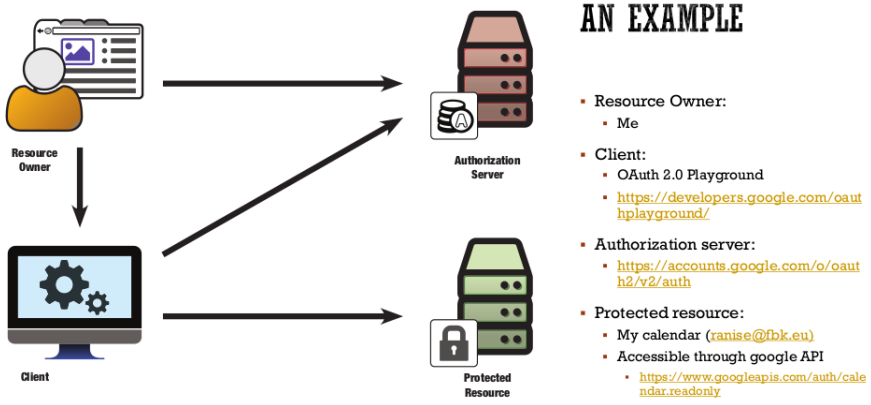
In many cases, authorization server and protected resource "live" on the same server, although this is not always the case.

37.1.1 OAuth tokens

Represent granted delegated authorities from the resource owner to the client for the protected resource. They are issued by the authorization server and are used by the client: format is opaque to clients. They are consumed by a protected resource. A token is not opaque to AS and RS, which to understand what is inside them use database lookup, put info in the token (JWT). RS query the AS.

37.1.2 Remarks

- Core protocol defined only for HTTP, relies on TLS for securing messages.
- It is not **an authentication protocol**: relies on authentication in several places (client authentication to token endpoint, resource owner authentication at authorization endpoint). Authentication protocols can be built using OAuth (OpenID Connect).
- It allows a user to delegate to a piece of software but not to another user.
- No authorization processing: tokens can represent scopes and other authorization information, processing of this information is up to the resource server. However, several methods (e.g., Json Web Token) are used to communicate this information.
- No token format: a token is opaque to the client and need to be issued by the authorization server and understood by the resource server, but they're free to use whatever format they want. JSON Web Tokens (JWT) provide a useful common format.
- There is **not** a single protocol: OAuth 2.0 is a framework consisting of several flows.



37.1.3 Key points

Example of Google OAuth 2.0 playground:

1. Need to log into the Provider's OAuth service when redirected
2. Explicit consent to limited access
   - **Scope**: string that represent what the token can do
     - Client can ask for scopes
     - Resource owner approves scopes
     - Access token is bound to scopes
   - Type of action (read, write, delete)
   - Type of resource (photos, metadata, profile)
   - Time of access (if the user is offline, there are limited number of accesses)
3. Protected Resource validates access token (i.e. requested access) when getting it from the Client. It needs a suitable cryptographic mechanisms to protect integrity and avoid forging!

Refresh tokens

When the user is no more there, access tokens still work. When the access token stops working the client uses expiration and revocation. To get a new token it is necessary to repeat the process of request (interactive grants: send the resource owner to the authorization endpoint). If the user is not available, it is also possible to ask **refresh tokens**, which are issued alongside the access token and used for getting new access tokens: they are presented along with client credentials, they are not good for calling protected resources directly).

Authentication codes

Why do clients need to obtain authorization codes before access tokens? Clients cannot be trusted (e.g., Man in the browser), APIs do not know who is calling them (bearer token): anyone presenting the access token will be granted access! To prevent this kind of problems, exchange is done server-to-server and requires both the client_id and client_secret, preventing even the (client) resource owner from obtaining the access token.

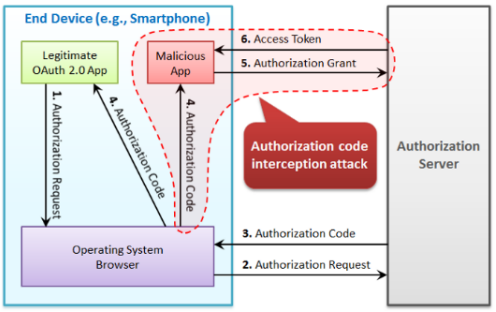
37.2 Authorization code flow

1. Client redirects the resource owner to the authorization server's authorization endpoint;
2. Resource owner authenticates to the authorization server;
3. Resource owner authorizes the client;
4. Authorization server redirects resource owner back to the client with an authorization code;
5. Client authenticates using its own credentials and sends the authorization code to the authorization server's token endpoint;
6. Authorization server issues an OAuth access token to the client;
7. Client accesses the protected resource using the access token.

37.3 Some attacks and mitigations

37.3.1 Authorization code intercept

The attacker manages to register a malicious application on the client device and registers a custom URI scheme that is also used by another application. The attacker has access to the OAuth 2.0 "client_id" and "client_secret". If the client is a mobile app, then nothing can be assumed to remain "secret".



Mitigation: **PKCE** (Proof Key Code Exchange)

1. Client creates a code challenge (type of challenge function is selected by the client, e.g., the hash of some unique secret) and sends it in the authorization request;
2. Authorization server authenticates the resource owner and returns the code;
3. In order to get the access token, client must send the code and prove that it initiated the flow. To do this, it sends the code verifier (e.g., unique secret from the first step) together with the authorization code;
4. Authorization server verifies whether the (hash of) code verifier matches the code challenge and returns the access token.

PKCE allows the legitimate app to create an ephemeral one-time secret and use that to authenticate to the Authorization server. A malicious app, even if it is able to steal the authorization code, will not have the one-time secret and so will be unable to trade the stolen code for the access token.

37.3.2 Attacks on JWT

JWT stands for JSON Web Token. It is the de facto format for OAuth 2.0 access token (standard does not impose any particular format). JWT consists of 3 components:

- **Header**: identifies algorithm used to sign;

```
{
 "alg" : "HS256",
 "typ" : "JWT"
}

base64url encoded string: eyBhbGcgOiBIUzI1NiwgdHlwIDogSldUIH0K
```

- **Payload**: information actually used for access control;

```
{
 "user_name" : "admin",
}

base64url encoded string: eyB1c2VyX25hbWUgOiBhZG1pbiB9Cg
```

- **Signature**: used to validate that the token has not been tampered with. Calculated by concatenating the header with the payload, then signing with the algorithm specified in the header.

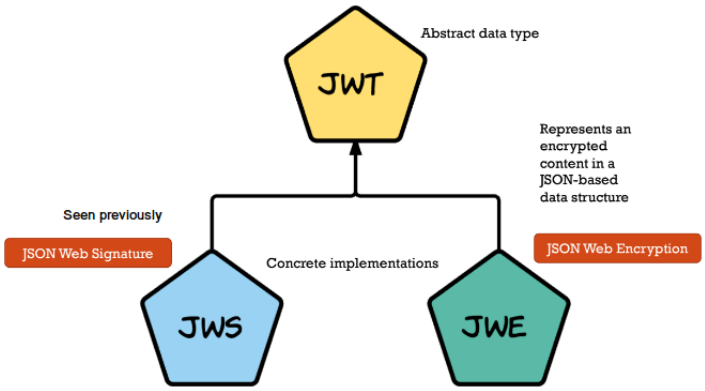
```
signature = HMAC-SHA256(base64urlEncode(header) + '.' +
base64urlEncode(payload), secret_key)

// Let's just say the value of secret_key is "key".

-> signature function returns
4Hb/6ibbViPOzq9SJflsNGPWSk6B8F6EqVrkNjpXh7M
```

The complete token is obtained by concatenating the 3 components above.

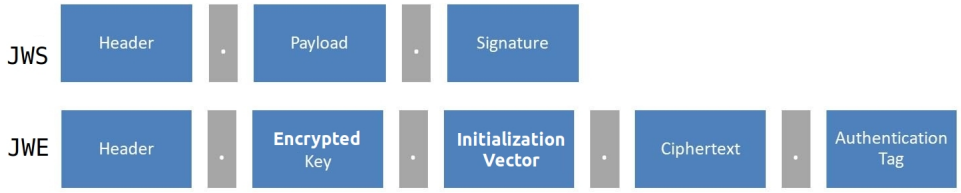
37.4 Digression on JOSE

JWT defines a container to transport data between interested parties; it became an IETF standard in May 2015 with the RFC 7519. There are multiple usages of JWT:

- propagate one's identity between interested parties;
- propagate user entitlements between interested parties;
- transfer data securely between interested parties over a unsecured channel;
- assert one's identity, given that the recipient of the JWT trusts the asserting party.



37.4.1 JWS vs JWE



37.4.2 JOSE (Javascript Object Signing and Encryption)

JOSE is a standard saying how to use cryptographic primitives to implement JWT (integrity protection and encryption). In March 2017, a serious flaw was discovered in many popular implementations of JSON Web Encryption (part of JOSE), the invalid curve attack (a vulnerability in the implementation of a primitive of ECC leads to an attack).

37.4.3 Bypassing JWT

When implemented correctly, JSON web tokens provide a secure way to identify the user since the data contained in the payload section cannot be tampered with: since users do not have access to the secret key, they cannot sign the token themselves. If implemented incorrectly, there are ways that an attacker can bypass the security mechanism and forge arbitrary tokens:

- Change the algorithm type
- Provide a non-valid signature
- Brute-force the secret key
- Leak the secret key
- Key ID manipulation

Change the alg type

If the application does not restrict the algorithm type used in the JWT, an attacker can specify which algorithm to use, which could compromise the security of the token.

- **None algorithm**: if the alg field is set to “None”, any token would be considered valid if their signature section is set to empty. It was originally used for debugging, if not turned off in production, it would allow attackers to forge any token they want by setting the alg field to "None"; they can then impersonate anyone on the site by using forged tokens.
- **Switching between RSA and HMAC**: with HMAC, the token would be signed with a key, then later verified with the same key. It is critical that the secret key for HMAC tokens is kept secret. With RSA, the token is first created with a private key, then verified with corresponding public key. Consider RSA scenario where tokens are signed with private key A and verified with private key B: if attackers change the alg to HMAC, they might be able to create valid tokens by signing the forged tokens with the RSA public key B. Originally when the token is signed with RSA, the program verifies it with the RSA public key B, when the signing algorithm is switched to HMAC, the token is still verified with the RSA public key B, but this time, the token can be signed with the same public key B.

Provide a non-valid signature

It is possible that the signature of the token is never verified after it arrives at the application, so an attacker could simply bypass the security mechanism by providing an invalid signature.

Brute force the secret key

The attacker knows a lot of information: algorithm used to sign the token, payload that was signed, and the resulting signature. If the key used to sign the token is not complex enough, guessing becomes possible.

Leak the secret key

If another vulnerability (like a directory traversal) exists that allows the attacker to read the file where the key value is stored, the attacker can steal the key and sign arbitrary tokens.

KID manipulation

KID stands for **Key Identifier**: it is a optional header field in JWTs allowing developers to specify the key to be used for verifying the token.

- **Directory traversal**: KID often used to retrieve a key file from the file system; if not sanitized before use, it can lead to a directory traversal attack. The attacker can force the application into using a publicly available file as the key, and sign an HMAC token using that file.
- **SQL injection**: KID could also be used to retrieve the key from a database; if SQL injection is possible on the KID parameter, attackers can use this injection to return any value they want.

37.4.4 More JWT security issues

- **Information leak**: if the token is not encrypted, anyone can base64 decode the token and read the token's payload. If the token contains sensitive information, it might become a source of information leaks. A properly implemented signature section of the JSON web token provides integrity not confidentiality.
- **Command injection**: when the KID parameter is passed directly into an insecure file read operation, it is possible to inject commands into the code flow. This can happen when an application passes any of the header parameters un-sanitized into any operating system function resembling system(), exec(), etc.

JSON web tokens are just another form of user input. They should always be handled with skepticism and sanitized rigorously.

37.4.5 More on JWTs

JWT are used by:

- OAuth 2.0 for authorization
- OpenID Connection for authentication (see later)
- Applications to keep track of state between requests
- Backend services to propagate authorization information in micro-service architectures

JWTs as OAuth 2.0 tokens

Traditional OAuth 2.0 tokens are bearer tokens. If one becomes compromised, it can be used without restrictions by whoever possesses it. It is strongly recommended to keep the lifetime of access tokens as short as possible: token lifetimes of minutes or hours are quite common, lifetimes of days or months are not recommended. If possible, short-lived access tokens should be combined with refresh tokens to improve security. Modern additions to the specification address the bearer token properties by introducing proof-of-possession mechanisms.

JWTs and Proof Of Possession

RFC 7800 describes how a JWT can declare that the presenter of the JWT possesses a particular proof-of-possession (PoP) key and how the recipient can cryptographically confirm proof of possession of the key by the presenter. Proof of possession of a key is also sometimes described as the presenter being a holder-of-key (analogous to a credit card protected with the owner's signature: even if someone steals a credit card he cannot use it without proving the ownership, proved by the signature).

37.5 OAuth 2.0 for authentication (bad idea)

Authentication tells an application who the current user is and whether or not he/she is present. Moreover refresh tokens can be used to obtain an access token **even when the user is not present**! The possibility to authenticate was excerpt from the OAuth 2.0 standard. Authenticating resource owners to clients is out of scope for this specification. The assumption that possession of a valid access token is enough to prove that a user is authenticated is true only in some cases (when the access token was freshly minted).

Why it is a bad idea

- **Access tokens as proof of authentication**: mere possession of an access token doesn't tell the client anything on its own. In OAuth, the token is designed to be opaque to the client, but in the context of a user authentication, the client needs to be able to derive some information from the token. The client is not the intended audience of the OAuth access token, it is the authorized presenter of that token, and the audience is in fact the protected resource;
- **Injection of access tokens**: a very dangerous threat occurs when clients accept access tokens from sources other than the return call from the token endpoint. This can occur if different parts of an application pass the access token between components in order to "share" access among them. This is problematic because it opens up a place for access tokens to potentially be injected into an application by an outside party (and potentially leak outside of the application). If the client application does not validate the access token through some mechanism, it has no way of differentiating between a valid token and an attack token. This can be mitigated by using the authorization code flow and only accepting tokens directly from the authorization server's token enpdoint, and by using a state value that is un-guessable by an attacker;
- **Injection of invalid user information**: if an attacker is able to intercept or coopt one of the calls from the client, it could alter the content of the returned user information without the client being able to know anything was amiss. This would allow an attacker to impersonate a user at a naive client by simply swapping out a user identifier in the right call sequence. This can be mitigated by getting the authentication information directly from the identity provider during the authentication protocol process (such as along side the OAuth token) and by protecting the authentication information with a verifiable signature;
- **Different protocols for every potential identity provider**: while the authorization may happen the same way at each provider, the conveyance of the authentication information could be different. This problem can be mitigated by providers using a standard authentication protocol built on top of OAuth so that no matter where the identity information is coming from, it is transmitted in the same way. The mechanisms for conveying authentication information discussed here are explicitly left out of scope for OAuth, which defines no specific token format, no common set of scopes for the access token, and does not at all address how a protected resource validates an access token.

The solution: **OpenID Connect**.

37.6 OpenID Connect

It is an open standard published in early 2014 that defines an interoperable way to use OAuth 2.0 to perform user authentication. Instead of building a different protocol to each potential identity provider, an application can speak one protocol to as many providers as they want to work with. OpenID Connect is built directly on OAuth 2.0 and uses the JSON Object Signing And Encryption (JOSE) suite of specifications for carrying signed and encrypted information around in different places. OpenID Connect manages to avoid many of the pitfalls discussed above by adding several key components to the OAuth base.

37.6.1 ID tokens

An ID token is a signed JWT given to the client application along side the regular OAuth access token. It contains a set of claims about the authentication session, including:

- an identifier for the user (**sub**),
- the identifier for the identity provider who issued the token (**iss**),
- the identifier of the client for which this token was created (**aud**).

Additionally, the ID Token contains information about the token's valid (and usually short) lifetime as well as any information about the authentication context to be conveyed to the client, such as how long ago the user was presented with a primary authentication mechanism. Since the format of the ID Token is known by the client, it is able to parse the content of the token directly and obtain this information without relying on an external service to do so. ID token is signed by the identity provider's private key, adding an additional layer of protection to the claims inside of it in addition to the TLS transport protection that was used to get the token in the first place, preventing a class of impersonation attacks. By applying a few simple checks to this ID token, a client can protect itself from a large number of common attacks.

37.6.2 User info endpoint

Clients are not required to use the access token, since the ID Token contains all the necessary information for processing the authentication event. For compatibility reasons, OpenID Connect always issues the ID token along side an OAuth access token. In addition to the claims in the ID Token, OpenID Connect defines a standard protected resource that contains claims about the current user. The claims here are not part of the authentication process, as discussed above, but instead provide bundled identity attributes that make the authentication protocol more valuable to application developers. The OpenID Connect scopes can be used along side other non-OpenID-Connect OAuth scopes without conflict, and the access token issued can potentially be targeted at several different protected resources: this allows an OpenID Connect identity system to smoothly coexist with an OAuth authorization system.

37.6.3 Dynamic server discovery and client registration

With OpenID Connect, a common protected API is deployed across a wide variety of clients and providers, all of which need to know about each other to operate. It would not be scalable for each client to have to know ahead of time about each provider, and it would be even more unscalable to require each provider to know about each potential client. OpenID Connect defines a discovery protocol that allows clients to easily fetch information on how to interact with a specific identity provider. On the other side of the transaction, OpenID Connect defines a client registration protocol that allows clients to be introduced to new identity providers. By using these two mechanisms and a common identity API, OpenID Connect can work at internet scale, where no parties have to know about each other ahead of time.