Chapter 12

Enforcing Security with Cryptography

12.1. Introduction

The world famous **I**nternational **O**rganization for **S**tandardization (ISO) defines in its norm ISO 27001:2005 (Information technology - Security techniques - Information security management systems - Requirements) the term " confidentiality " as follows. *Confidentiality is a characteristic that applies to information. To protect and preserve the confidentiality of information means to ensure that it is not made available or disclosed to unauthorized entities. In this context, entities include both individuals and processes.*

One way to ensure a high level of confidentiality should be to use some private communication network, with native devices of information protection. For instance some privately operated optical fiber network between two buildings of a financial institution. Nevertheless the cost for establishing and maintening of such networks is clearly not compatible with scaling. What is the alternative ? *Cryptography*. What is cryptography ? A collection of involved mathematical notions, miscellaneous engineering designs and a large amount of daily used software that allow thousands of people per hour to buy or sell many articles on the Internet.

The very objective of cryptography is to allow confidential communications between two entities, namely human beings, computers or any processes, through a public network. By public network is meant an unrestricted communication medium

Chapter written by Sami HARARI and Laurent POINSOT.

with no access control such as the telephone network or Internet for instance. Cryptographic devices should make it impossible, if not to collect, to use informations illegitimately " sniffed " on the public network.

When confidentiality must be guaranteed, for instance for military messages or financial transactions, it is fundamental to employ cryptography. Nevertheless such techniques may be successfully used to achieve other security requirements such as integrity, authenticity and non-repudiation. Even if we give some account for the previous ideas, this chapter is mainly devoted to the protection of confidentiality.

From a very general point of view, a cryptographic kind of communication between two individuals (or computers or processes), say Alice and Bob, is composed in the following way. Before sending its information (on the public network) to Bob, Alice modifies it, by using a *cryptographic system* (a *cryptosystem* for short), into a new message called *ciphertext* or *cryptogram* which has the essential property to dissimulate the very nature of the original message, called *cleartext* or *plaintext*, to every entity other than Bob. This last one, the legitimate recipient of Alice's message, retrieves and decrypts the ciphertext in order to recover the original information of Alice. Since the network is public and, as so, freely accessible, any person may be able to intercept the ciphertext. However, because it is encrypted *for* Bob, this message seems to be without meaning and in fact unusable. In this way, in principle, the requirement of confidentiality is ensured.

All cryptosystems are quite similar in form and in principle, and they all share same operating process and fulfill similar tasks. Therefore the first part of this chapter is devoted to the general description of these common features: an accurate definition for cryptosystems will be given as well as a description of high level functionalities provided by cryptosystem devices. Furthermore the very existence of cryptography is related to threats on communication media; this is the reason why the concept of cryptanalysis is also introduced in the first part. Cryptosystems are classified in one or the other of the following two species: symmetric (or secret-key) cryptosystems, and public-key cryptosystems. A part of this chapter is dedicated to both families. In Section 12.3 we deal with the former; we provide a general description for secret-key cryptosystems, and we deal with some famous algorithms, namely DES, IDEA and AES, in order to illustrate the development of mathematical technologies in this area. The world famous RSA algorithm, as a relevant instance of public key cryptosystems, is detailed in Section 12.4. Then we stress the fundamental role played by prime numbers in asymmetric encryption: different techniques to prove primality or to provide prime numbers are presented.

12.2. Cryptography: From a General Perspective

In this section are introduced general definitions and high-level principles available for the remainder of this chapter.

12.2.1. *Cryptosystems*

The notion of cryptographic systems, although quite imprecise in appearance, may be given a rigorous mathematical definition (which is inspired from [STI 06]). Mathematically speaking, an *enciphering algorithm*, also called *cryptosystem* or *cipher system* [1], can be described as a collection of three non-empty sets, $\mathcal{P}$, $\mathcal{C}$ and $\mathcal{K}$ (in general these sets are finite), called sets of *plaintexts*, *ciphertexts* and *keys*, and two functions [2] $E\colon \mathcal{K} \to \mathcal{C}^{\mathcal{P}}$, that maps a key $k \in \mathcal{K}$ to a *enciphering* (or *encryption*) *function* $E_k : \mathcal{P} \to \mathcal{C}$, and $D\colon \mathcal{K} \to \mathcal{P}^{\mathcal{C}}$, that associates with every $k \in \mathcal{K}$ its *deciphering* (or *decryption*) *function* $D_k : \mathcal{C} \to \mathcal{P}$, which are required to satisfy a *decryption rule*: for every plaintext $x \in \mathcal{P}$,

$$D_k(E_k(x)) = x\ .$$

This rule is fundamental for the decryption process, and more precisely to make such a process possible. Indeed, let us assume that $y \in \mathcal{C}$ is the ciphertext $E_k(x)$ for some plaintext $x \in \mathcal{P}$ (and key $k \in \mathcal{K}$). The decryption rule points out that we can recover this plaintext from the ciphertext by an application of the decryption function D_k: $D_k(y) = D_k(E_k(x)) = x$. While quite simple, this is the most important feature of a cryptosystem, and this leads to the search for invertible functions [3] E_k and D_k (for every key k). This is the reason why we will present some cryptographic algorithms emphasizing the property of invertibility satisfied by the encryption/decryption functions.

From this formalism we deduce that Alice must know the map E_k while D_k has to be known by Bob in order to make possible the decryption process. Besides Bob, if someone else is acquainted with the use of D_k, then he is able to decrypt any message enciphered by E_k, and the protection of confidentiality probably fails (except if the person under consideration is Alice!).

1. In the sequel we will freely use the terms " cryptographic " or " cipher " or " encryption " in order to avoid monotony.

2. Recall that the set of all maps from X to Y is usually denoted Y^X.

3. The decryption rule only implies that for each $k \in \mathcal{K}$, D_k is onto and E_k is one-to-one. Nevertheless, when $\mathcal{P}$ and $\mathcal{C}$ are finite with the same cardinal number, both maps are invertible (we also say that they are " bijective " or " one-one correspondences ").

Example 12.1 The following (secret-key) cryptosystem, called *one-time pad*, was invented by G.S. Vernam in 1917 (while published in 1926 in [VER 26]). Let us denote by $\mathbb{Z}_2$ the set $\{0,1\}$ of bits and let $\oplus$ be the addition of bits modulo two, also called *exclusive or* (shorter: *XOR*), given by the following (addition) table:

$\oplus$	0	1
0	0	1
1	1	0

Vernam's cryptosystem is formalized in the following fashion: $\mathcal{P} = \mathcal{C} = \mathcal{K} = (\mathbb{Z}_2)^\ell$ where ℓ is a positive integer. Therefore plaintexts, ciphertexts and keys are are ℓ-tuples of bits. For each key $k = (k_1, \dots, k_\ell)$ (where each k_i is a bit) the encryption is defined

$$\begin{array}{rlll} E_k : & (\mathbb{Z}_2)^\ell & \to & (\mathbb{Z}_2)^\ell \\ & x = (x_1, \dots, x_\ell) & \mapsto & x \oplus k = (x_1 \oplus k_1, \dots, x_\ell \oplus k_\ell) \ . \end{array}$$

So the ciphertext $E_k(x)$ corresponding to the plaintext x is equal to the component-wise modulo-two sum, which, by abuse, is also called *XOR* (or *exclusive or*), of x and k. The decryption function D_k is equal to E_k. It is quite easy to check that the decryption rule is satisfied. First of all, let us notice that for every $x, y \in (\mathbb{Z}_2)^\ell$ it holds that $(x \oplus y) \oplus y = x$ (this is due to the definition of $\oplus$ at the bit level). It follows that

$$\begin{array}{rcl} D_k(E_k(x)) & = & D_k(x_1 \oplus k_1, \dots, x_\ell \oplus k_\ell) \\ & = & ((x_1 \oplus k_1) \oplus k_1, \dots, (x_\ell \oplus k_\ell) \oplus k_\ell) \\ & = & (x_1, \dots, x_\ell) \\ & = & x \ . \end{array}$$

In order to illustrate the encryption process, let us suppose that $\ell = 4$, $x = (0,1,1,0)$, and $k = (1,1,0,1)$. Then $E_k(x) = (0 \oplus 1, 1 \oplus 1, 1 \oplus 0, 0 \oplus 1) = (1,0,1,1)$.

12.2.2. *Two Dissimilar Worlds*

As announced in the Introduction, there are two principal classes of cryptosystems, distinguished by the management of the secret on E_k and D_k.

Conventional, *symmetric* or *secret-key cryptosystems* are the encryption schemes where nobody knows the key k used to communicate, except the legitimate correspondents, say Alice and Bob. In this context, k is called the *secret key*. Functions E_k and D_k are secret quantities shared by the two interlocutors. In order to use such a cryptosystem Alice and Bob need to choose together the secret key, or at least one of them determines then communicates it to the other. In short Alice and Bob must agree on the choice of the secret key *before* any encrypted communication. In order to make this choice, they must meet physically in a secure area, or use a private network. The one-time pad of example 12.1, and also DES, IDEA and AES (described later) are secret key cryptosystems.

The other main class of encryption processes is given by the so-called *asymmetric* or *public-key cryptosystems*. The key k and the decryption function D_k are secret quantities only known by the receiver of confidential messages, Bob, while the encryption function E_k (and not the key k) is published by Bob (on his web page for instance) so that everybody who wishes to communicate with him can use it. In this situation Bob is the unique individual able to decrypt messages $E_k(x)$ since D_k is its own secret. The different roles played by the public E_k on one side and the secrets k, D_k on the other side, justify the term " asymmetric " for such cryptosystems; obviously " public-key " comes from the existence of this public quantity E_k. The RSA algorithm belongs to this class of algorithms.

We emphasize the fact that both classes of cryptosystems are based on very different mathematical techniques: invertible functions over some algebraic structures, probability theory and statistical analysis usually occurred in conventional cryptography, while prime numbers, computability and complexity theories are the main ingredients of the mathematical foundation for asymmetric encryption schemes.

12.2.3. *Functionalities Provided by Cryptographic Devices*

The application of cryptographic tools is not restricted to the protection of information confidentiality. It is actually possible to define four primitive functionalities provided by encryption devices: confidentiality, authenticity, integrity and non-repudiation. Each of them represents a mean of defense against a particular kind of threat. We review in a few words these cryptographic characteristics.

Confidentiality means that an information, after encryption, loses all meaning for all people except the legitimate protagonists of a cryptographic communication. An enemy that intercepts the plaintext must be unable to decrypt it for confidentiality to be preserved.

Integrity: In every communication (encrypted or not) it is expected the message to be received with no modifications, exactly as it was sent. Moreover, if a received message is different from the transmitted message, then the receiver must be able to detect it. We say that " integrity of messages against modifications is ensured " if the preservation of these two properties is secured.

Authenticity: The mission assigned to authentication consists in the guarantee that the received message comes from the entity (human, computer, process) which is supposed to send it. If an enemy – playing the role of Alice – sends a message to Bob, then the authenticity of the message must be questioned. Otherwise, Bob, believing the enemy is Alice, would send confidential information to him.

Non-repudiation is the means that avoid a receiver of a message to deny its transmission. This a fundamental protection for instance in the context of financial transactions.

In this chapter we only deal with integrity which is the heart of cryptography, and we do not develop the other cryptographic notions.

12.2.4. *Cryptanalysis: the Dark Side of Cryptology ?*

In the world of cryptography two kinds of entities coexist: the legitimate players of a enciphered communication, Alice and Bob, and an adversary (also called cryptanalyst, enemy, opponent, attacker) who tries to discover the key used to encipher; if he succeeds in this attempt, then the enemy has " broken " the cryptosystem: the *cryptanalysis* is successful.

Cryptography and cryptanalysis form the two sides of cryptology, the *science of secret*. In appearance, but only in appearance, the dark side of cryptology is cryptanalysis. Nevertheless this notion is also used to design systems to be invulnerable against some classes of cryptanalysis. Then it becomes essential to define models of the strength of an attacker so as to measure how strong the cryptosystem is. At this step *Kerckhoffs' principle* is often assumed. This assumption – defined by A. Kerckhoffs – means that the encryption algorithm is known by the enemy ([KER 83a, KER 83b]).

There exists a very basic cryptanalysis for every cryptosystem, called *brute-force attack*, which in theory should be able to break any encryption algorithm. It is not sophisticated at all since it consists in trying to decrypt a ciphertext with all possible keys until an understandable plaintext is obtained. An adversary will find the key after an average of $\frac{|\mathcal{K}|}{2}$ attempts[4].

The number $|\mathcal{K}|$ of possible keys is clearly a fundamental parameter to measure how strong a cryptosystem is, with respect to a brute-force attack. Modern cryptosystems with a size of at least 160 bits to encode a key are considered secure against this

4. The quantity $|X|$ is the number of elements or *cardinal* of the finite set X.

trivial attack because, even for very advanced computers, an exhaustive search in a set of 2^{159} seems to be impossible in practice. Usually, an attack is considered successful when it requires less time to get the key than a brute-force attack.

Obviously more sophisticated cryptanalysis may be encountered. Their common goal is always to find the key used to encrypt messages. The most common types of attacks are classified by increasing order of adversary's power. The list is given below.

Known-ciphertext attacks. The adversary is assumed to have only access to a set of ciphertexts (from unknown plaintexts and a fixed unknown key);

Known-plaintext attacks. The enemy has samples of both the plaintext and its encrypted version (by a given and unknown key), the ciphertext, and is free to make use of them to reveal the key;

Chosen-plaintext attacks. This mode presumes that the attacker has the capability to choose arbitrary plaintexts to be encrypted and obtain the corresponding ciphertexts;

Chosen-ciphertext attacks. The opponent collects information by choosing one or several ciphertexts and obtaining their decryption under an unknown key.

This classification allows us to define several degrees of cryptographic resistance. For instance it is possible to prove that the one-time pad is invulnerable with respect to a known-ciphertext attack while it is easily broken by a known-plaintext attack: let us assume that a plaintext m and its ciphertext $c = m \oplus k$ are known, then the key is immediately found by computing $m \oplus c = k$.

12.2.5. *General Requirements to Avoid Vulnerabilities*

There are three theoretical models to measure the level of security of a cryptographic device. In 1949, Claude Shannon, founder of modern cryptography, gave the mathematical bases of contemporary cryptology in his famous article [SHA 49]. In this paper he introduced the two first criteria for a cryptosystem to be secure: *unconditional security* and *statistical security*.

A cryptosystem is said to provide *unconditional security* when any kind of knowledge on a ciphertext does not reveal any information about the corresponding plaintext. As an example we can prove that for Vernam's cryptosystem such a cryptographic property holds whenever a new randomly chosen key is used for each encryption. This very strong feature ensures invulnerability against every known-plaintext attack.

Nevertheless, one-time pad, as all secret-key algorithms, involves a key exchange among the legitimate interlocutors. But to satisfy unconditional security they are forced to use a new key for each of their confidential communications. We easily see the limitation of such a process in practice. In order to get around it, Shannon defined an other resistance criterion, namely *statistical security* that is based on two more fundamental properties called *diffusion* and *confusion*.

Under the name of *diffusion* Shannon defined the fact that every letter (or more generally symbol) of a ciphertext should be dependent of every letter of the corresponding plaintext and of the key. The goal of this is the following: two ciphertexts, where one of them is due to a modification – even a minimal one – of the plaintext or of the key, must be very different. It is a kind of dependence of ciphertexts to initial condition (plaintext or key used). A slight difference at the input of a cryptosystem must produce a large difference in its output.

Confusion refers to making the relationship between the key and the ciphertext as complex and involved as possible in order to hide any statistical structures that would be used to discover bits of information from the plaintext without knowledge of the key. For instance, statistics of natural languages must be destroyed during the encryption process so that they become inpractical for an adversary. We will observe soon that these two properties, diffusion and confusion, establish the architectural pattern of modern symmetric encryption algorithms.

The last approach to cryptographic security, called *computational security*, introduced by Whitfield Diffie and Martin Hellman in their joint-work [DIF 76], only concerns public-key encryption schemes. Such an algorithm is said to provide *computational security* if the best known attack requires too many computations to be really feasible in practice. In general we prove that breaking a cryptosystem is equivalent to solve a problem known to be difficult in the sense that the construction of an explicit solution is impossible in practice (but not in theory!). Notice also that a computationally secure scheme is not unconditionally secure.

12.3. Symmetric Encryption Schemes

This section is devoted to symmetric encryption schemes: the high-level design is presented at first, followed by famous instances of such schemes.

12.3.1. *The Secret Key*

Let us briefly recall how a secret-key algorithm is implemented. For a symmetrically ciphered communication, Alice and Bob, and no other entity, have the common secret key. Thus Alice encrypts her message with this key, and sends to Bob, who can recover the original message from the ciphertext he received by using the key. Even if the cryptosystem used is known by everybody – in accordance with Kerckoffs' principle – an adversary cannot decrypt any intercepted confidential message since he does not possess the key.

The choice of the key by Alice and Bob is a tricky problem. Indeed, either they physically meet or one of them sends the key to the other using a communication network secured in some way, for instance a private optic fiber between two buildings, or by the use of a key-exchange protocol. This problem is not treated in this chapter. See [MEN 97], available on-line at `http://www.cacr.math.uwaterloo.ca/hac/`, for a good reference on key-exchange protocols.

12.3.2. *Iterated Structures and Block Ciphers*

For evident sake of efficiency, encryption processes are performed by computers. Thus, the messages (plain or cipher) are treated as blocks of bits (or bytes) built following an *iterated* architecture that allows a high level of confusion and diffusion. An internal *round function* T is used. It takes two arguments: a message m and a *secret-subkey* or *round subkey* k (both are blocks of bits). The subkey is produced from the secret key, called *master key*, by some *derivation algorithm*. Even though they are required for a symmetric encryption scheme, these algorithms are not treated in more detail in this chapter.

The round function is required to satisfy the following property to make decryption possible. With a fixed round subkey k, the function $T_k \colon m \mapsto T(m, k)$ must be invertible. This is actually the realization of the decryption rule in this particular context. The argument m is called *round plaintext* and $T(m, k)$ is the *round ciphertext*. The round function consists in a sequence of complex mathematical transformations in order to make its result $T(m, k)$ unintelligible. More precisely, T must implement confusion and diffusion requirements. In particular an output block of such a function must be dependent of an important number (at least half the number) of bits of plaintext and round subkey.

In order to confuse and diffuse, the round function is iterated some number r of times as follows. Let m be the message to encrypt. The following sequence of computations is done.

$$\begin{array}{rcl} m_0 & = & m; \\ m_{i+1} & = & T_{k_{i+1}}(m_i) \text{ for } 0 \leq i \leq r-1 \end{array}$$

where k_i denotes the subkey related to the ith round. The ciphertext c, obtained as output of the last round, is given by formulae:

$$\begin{aligned} c &= m_r \\ &= T_{k_r}(m_{r-1}) \\ &= T_{k_r} \circ T_{k_{r-1}} \circ \cdots \circ T_{k_2} \circ T_{k_1}(m) \,. \end{aligned}$$

where " $\circ$ " is the usual composition of functions. This iterated architecture turns out to be unavoidable to obtain convenient levels of confusion and diffusion in order to ensure statistical security. More precisely, iteration increases the diffusion.

Let us take a look at the deciphering process. Recall that for a given round subkey k, the function T_k is required to be invertible, which implies that there exists a map T_k^{-1} such that for any block x, $T_k^{-1}(T_k(x)) = x$. Decryption is performed by " reversing the time ". More precisely, it is done by replacing the round function T_k by its inverse T_k^{-1}, and running in the sequence of subkeys in the reverse order. Formally from the ciphertext c the plaintext m is obtained by:

$$\begin{aligned} c_0 &= c; \\ c_{i+1} &= T_{\widehat{k}_{i+1}}^{-1}(c_i) \text{ for } 0 \leq i \leq r-1 \end{aligned}$$

where $\widehat{k}_i$ denotes the subkey k_{r+1-i} related to the $r+1-i$th round so that:

$$\begin{aligned} \widehat{k}_1 &= k_r \\ \widehat{k}_2 &= k_{r-1} \\ &\dots \\ \widehat{k}_r &= k_1 \,. \end{aligned}$$

According to the invertibility of the round function (with a fixed round subkey), the final block c_r is clearly equal to the original plaintext m.

12.3.3. *Some Famous Algorithms: a Short Story of the Evolution of Mathematical Techniques*

Most of the famous symmetric encryption schemes make use of an iterated structure with some possible minor modifications at the first and final rounds. Therefore such cryptosystems only differ from the point of view of the size of data (plain and ciphertext, secret key, subkey), of the number of rounds, of the internal round function and the derivation algorithm used. In what follows three of the most renowned secret-key ciphers, namely DES, IDEA, and AES, are described insisting on the evolution of mathematical constructions in use in those algorithms.

The Seventies: DES - Data Encryption Standard

Data Encryption Standard, DES, was designed by IBM during the seventies, and became an encryption standard in 1977 for United States of America's official documents. Its status as a standard – for five years – was evaluated several times: the last time being in 1999.

In [FIP 99] the DES is completely described and in [FIP 87] its different operation modes are presented. This symmetric algorithm operates on 64-bits plaintexts, ciphertexts and secret-keys. Actually, 8 bits from the key are parity bits: the eighth bit of each byte of the key takes the value such that the number of bits equal to 1 in this byte is an even number. A subkey (for a given round) is given by 48 bits of the master key – except parity bits – in some specific order. The ciphertext is obtained after sixteen rounds.

Let us study the DES round function. It is formally defined as a *Feistel structure* or *Feistel scheme* named after the American cryptographer Horst Feistel [FEI 73]. In such a scheme, blocks have an even number 2ℓ of bits ($\ell = 32$ in the case of DES). The first ℓ consecutive bits of some block B are denoted, as a block, by L, while R is given by B's last ℓ bits in such a way that $B = (L, R)$. Let f be a function that takes two blocks as input, the first block having length ℓ. This function produces as output also a block of size ℓ. The round function T for the Feistel structure associated with f operates as follows: it takes $B = (L, R)$ and a round subkey k as entries, it flips L and R, and transforms L into $f(R, k) \oplus L$. This can be written in a mathematical form:

$$\begin{aligned} T(B,k) &= T((L,R),k) \\ &= (R, f(R,k) \oplus L). \end{aligned}$$

It can be easily checked that given any map f as above, the round function T, with a fixed round key k, is invertible. This is an essential property for the deciphering process in such a cryptosystem. Let us prove this property. We define the map $U_k(L, R) := (f(L, k) \oplus R, L)$ which will be shown to be the inverse of $T_k \colon (L, R) \mapsto T((L, R), k)$. Notice that if we denote by σ the permutation $\sigma(L, R) = (R, L)$, then $U_k = \sigma \circ T_k \circ \sigma$. Moreover $U_k(T_k(L, R)) = (L, R)$. Indeed let us define $L' = R$ and $R' = f(R, k) \oplus L$.

$$\begin{aligned} U_k(T_k(L,R)) &= U_k(R, f(R,k) \oplus L) \\ &= U_k(L', R') \\ &= (f(L',k) \oplus R', L') \\ &= (f(R,k) \oplus (f(R,k) \oplus L), R) \\ &= (L, R). \end{aligned}$$

As a result, such a Feistel scheme may be used as a round function in an iterated symmetric encryption algorithm.

In order to complete the description for the DES round function, a description of the function f used in this system is needed. This is an important function because confusion is based on it, while diffusion is obtained by the iterated structure itself. The function f takes as its first argument a block of size 32 (the 32 first or last bits of the block to encrypt) which is denoted by X. The second argument is a subkey, so here a block, say Y, of 48 bits. The result $f(X, Y)$ is a block of 32 bits (according to the specifications of Feistel structures). The function f carries out a computation in four steps:

1) X is transformed by a function E, that takes 32 bits in input and produces a block of 48 bits, in such a way that $E(X)$ consists in the bits of X in an other order where sixteen of them are duplicated. More precisely the 48 bits of $E(X)$ are obtained by selecting the bits of X according to the order induced by the following table:

Function E

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

Thus, as an example, the 4 first bits of $E(X)$ are the bits 32, 1, 2 and 3 of X whereas the 3 last bits are the bits 31, 32 and 1 of X;

2) The result $E(X) \oplus Y$ is then computed and written as a concatenation of 8 subblocks, each of them consisting of 6 bits:

$$E(X) \oplus Y = B_1 B_2 B_3 B_4 B_5 B_6 B_7 B_8$$

where for each $i \in \{1, \dots, 8\}$, B_i has a length of 6 bits;

3) For each $i = 1, \dots, 8$, B_i goes through a function S_i, called a *substitution* or an *S-box*. Such a box takes 6 bits in input and gives 4 bits as output. The result of this step is given by the concatenation of the $S_i(B_i)$, *i.e.*, the block of 32 bits:

$$S = S_1(B_1)S_2(B_2)S_3(B_3)S_4(B_4)S_5(B_5)S_6(B_6)S_7(B_7)S_8(B_8).$$

Each S-box S_i is represented by a table with 4 rows and 16 columns. Its rows are indexed from the top to the bottom with integers from 0 to 3 and its columns from the left to the right by integers from 0 to 15. Each entry contains an integer between 0 and

15. For instance, the S-box S_1 is given by the following table:

S_1

14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

Let us see the action of an S-box, say S_i, on a block B_i of 6 bits. The first and last bits of B_i are interpreted as a binary representation of an integer, say a, between 0 and 3. The four other bits represent a binary representation of an other integer, say b, between 0 and 15. The entry (a, b) of the table associated with S_i, *i.e.*, the integer given at the intersection of the ath row and the bth column, may be written as a block of 4 bits in its binary representation (since, by definition, it is an integer between 0 and 15). This block is taken as the output of S_i, or in other terms, the value $S_i(B_i)$. For instance, let B_1 be the block 011011. The corresponding index for the row of S_1 is represented by 01 so it is equal to $0 \times 2^1 + 1 \times 2^0 = 1$ in decimal representation. The corresponding index for the column of S_1 is given by 1101 which is the binary representation of $1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 13$. Therefore $S_1(B_1)$ is the binary representation of the integer 5 given as the entry $(1, 13)$. Since 5 is represented by 0101, the result is $S_1(B_1) = 0101$.

These S-boxes are nonlinear in the sense that in general $S_i(B_i \oplus B_i') \neq S_i(B_i) \oplus S_i(B_i')$. They destroy the algebraic structure, therefore produce confusion for this cryptogram;

4) At the input of these eight S-boxes we have a block S of 32 bits. The last step of internal computations of f is a re-ordering of these bits using a permutation P. It is represented in the table below:

Permutation P

16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

The output $P(S)$ is obtained from S by taking the sixteenth bit of S for the first bit of $P(S)$, the seventh bit of S for the second bit of $P(S)$, *etc.* (at the end, the twenty-fifth bit of S is used as the thirty-second bit of $P(S)$). $P(S)$ is taken as the result of $f(X, Y)$ for the round function.

In order to summarize this situation, to compute $f(X,Y)$, $B_1,\ldots,B_8$ are defined as blocks of 6 bits each by

$$B_1B_2\ldots B_8 = E(X)\oplus Y$$

then the block $f(X,Y)$ is defined by

$$f(X,Y) = P(S_1(B_1)S_2(B_2)\ldots S_8(B_8)).$$

The DES round function is now fully described. We are in position to conclude with the presentation of the encryption process by the DES algorithm. An initial step, before the 16 rounds, is applied to the block that represents the plaintext: it goes through a permutation IP, called *initial permutation*, the operation of which is given by the following table.

Permutation IP

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	22	14	6
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Therefore the permuted block has bit number 58 from the original block as its first bit, then bit 50 for its second, and so on. This initial step is followed by the sixteen iterations of the round function. Finally if (L_{16}, R_{16}) denotes the 64-bits blocks produced at the sixteenth, and last, round, then the encryption process is ended by applying to $\sigma(L_{16}, R_{16}) = (R_{16}, L_{16})$ the inverse IP^{-1} of IP.

We are now in position to present this algorithm in a more compact way. Let k be the master key, and k_i be the subkey from round number i. Let $E_k^{\mathbf{DES}}$ be the DES encryption function. The ciphertext $E_k^{\mathbf{DES}}(m)$ of a plaintext m is computed as follows.

$$\begin{array}{lcl} (L_0, R_0) & = & IP(m); \\ (L_{i+1}, R_{i+1}) & = & T_{k_{i+1}}(L_i, R_i) \text{ for } i = 0,\ldots,15; \\ E_k^{\mathbf{DES}}(m) & = & IP^{-1}(R_{16}, L_{16}). \end{array}$$

Using a more condensed notation:

$$E_k^{\mathbf{DES}}(m) = \left(IP^{-1}\circ\sigma\circ T_{k_{16}}\circ\cdots\circ T_{k_1}\circ IP\right)(m)\ .$$

Notice that the final permutation is not applied to (L_{16}, R_{16}) but to $\sigma(L_{16}, R_{16})$, *i.e.*, (R_{16}, L_{16}). Since this permutation, IP^{-1}, is the inverse of IP, in order to perform decryption the same algorithm is applied on $E_k^{\mathbf{DES}}(m)$, subkeys being used in a reverse order from k_{16} to k_1. In other terms, the decryption function is defined by

$$D_k^{\mathbf{DES}}(c) = (IP^{-1} \circ \sigma \circ T_{k_1} \circ \cdots \circ T_{k_{16}} \circ IP)(c) \ .$$

In order to check the decryption rule, we only need to notice that $T_k^{-1} = \sigma \circ T_k \circ \sigma$, and for every (L, R), $(\sigma \circ \sigma)(L, R) = (L, R)$. Therefore,

$$\begin{array}{l} D_k^{\mathbf{DES}}(E_k^{\mathbf{DES}}(m)) = \\ (IP^{-1} \circ \sigma \circ T_{k_1} \circ \cdots \circ T_{k_{16}} \circ IP)(E_k^{\mathbf{DES}}(m)) = \\ IP^{-1} \circ (\sigma \circ T_{k_1} \circ \sigma) \circ \cdots \circ (\sigma \circ T_{k_{16}} \circ \sigma) \circ \sigma \circ IP(E_k^{\mathbf{DES}}(m)) = \\ IP^{-1} \circ T_{k_1}^{-1} \circ \cdots \circ T_{k_{16}}^{-1} \circ \sigma \circ IP(E_k^{\mathbf{DES}}(m)) = \\ IP^{-1} \circ T_{k_1}^{-1} \circ \cdots \circ T_{k_{16}}^{-1} \circ \sigma \circ IP \circ IP^{-1} \circ \sigma \circ T_{k_{16}} \circ \cdots \circ T_{k_1} \circ IP(m) = m \\ \text{(eliminating consecutive compositions of a map and its inverse).} \end{array}$$

Document [FIP 99] also contains the description of an other algorithm for symmetric encryption, TDEA (for Triple Data Encryption Algorithm), called *triple DES*. It is defined as an iteration of the original DES. Let $k^{(1)}$, $k^{(2)}$ and $k^{(3)}$ be three master keys subject to particular independence properties (given in [FIP 99]). Let m be a 64 bits long block to encode.

1) Encryption algorithm: block m is transformed into a new block c (64 bits) as follows:

$$c = E_{k^{(3)}}^{\mathbf{DES}}(D_{k^{(2)}}^{\mathbf{DES}}(E_{k^{(1)}}^{\mathbf{DES}}(m)));$$

2) Decryption algorithm: m is recovered from the ciphertext c by computing:

$$m = D_{k^{(1)}}^{\mathbf{DES}}(E_{k^{(2)}}^{\mathbf{DES}}(D_{k^{(3)}}^{\mathbf{DES}}(c))) \ .$$

The Nineties: IDEA - International Data Encryption Algorithm

IDEA algorithm, invented by Xuejia Lai and James L. Massey, is described in [LAI 90] and [LAI 92].

IDEA was explictly designed to fulfill confusion and diffusion requirements. Like DES, it is based on an iterated structure. However the method used to produce invertible functions – in order to make possible the decryption process – is not based on Feistel structures. IDEA round function relies on more involved mathematical structures, namely the *groups*. An *internal composition law*, denoted by $*$, on a set E is a function that associates an ordered pair (x, y) of members of E to some z that belongs to E: we denote this z by $x * y$. A *group* is then defined as a non-empty set G together with an internal composition law that satisfies the following axioms:

1) Associativity: for every x, y, z in G, $x * (y * z) = (x * y) * z$;

2) Neutral element: there is some $e \in G$ such that for every $x \in G$, $x * e = e * x = x$;

3) Inversion: for every $x \in G$, there is a unique $y_x \in G$ such that $x * y_x = y_x * x = e$. This element y_x is usually denoted by x^{-1}.

For instance if p is a *prime* number – that is a positive integer > 1 with 1 and the number itself as only divisors (such that $2, 3, 5, 7, 11$, *etc.*) – then modulo p multiplication of positive integers is an internal composition group law on the set $\{1, 2, \cdots, p-1\}$. Similarly for every positive integer n, the set $\{0, \cdots, n-1\}$ becomes a group under modulo n addition. Finally, the set of all blocks of n bits with bit-wise modulo 2 sum, that is XOR, is another example of a group. IDEA is precisely based on these three algebraic structures.

In order to describe IDEA round function, the following notations will be used. Let n be an integer so that $2^{2^n} + 1$ is a prime number (for instance $n = 1$ or $n = 2$ or $n = 16$).

– As usually the symbol " $\oplus$ " is used to denote XOR operation between two blocks of 2^n. For instance with $n = 2$, $(0,1,1,0) \oplus (1,1,0,1) = (1,0,1,1)$;

– Each 2^n-bit long block can be identified with a unique integer between 0 and $2^{2^n} - 1$ written in binary representation. More generally, let us assume given a ℓ-bit block $(x_{\ell-1}, x_{\ell-2}, \cdots, x_1, x_0)$, $x_i \in \{0,1\}$. It represents the integer $x = \sum_{i=0}^{\ell-1} x_i 2^i$, and satisfies $0 \leq x \leq 2^\ell - 1$. It is thereby possible to compute a modulo 2^{2^n} addition under this identification (take $\ell = 2^n$). This operation is denoted by " $\boxplus$ ". For $n = 2$ so that $2^{2^n} = 16$, $(0,1,1,0)$ represents the integer 6, and $(1,1,0,1)$ the integer 13. Addition modulo 16 of 6 and 13 is, in binary notation, is equal to $(0,0,1,1)$. Therefore $(0,1,1,0) \boxplus (1,1,0,1) = (0,0,1,1)$;

– Each 2^n-bit long block, such that at least one of its bits is not zero, represents a unique integer between 1 and $2^{2^n} - 1$. The block, given by 2^n bits equal to zero, is declared to represent the integer 2^{2^n}. Since $2^{2^n} + 1$ is assumed to be prime, the set $\{1, 2, \ldots, 2^{2^n}\}$, under modulo $2^{2^n} + 1$ multiplication of integers, is a group. According to this identification between blocks and integers, we can apply this product, denoted by " $\odot$ ", to any two blocks (each of them composed of 2^n bits). For instance, $(0,1,1,0) \odot (1,1,0,1) = (1,0,1,0)$ since 6×13 is equal to 10 modulo $2^4 + 1 = 17$, and 10 is represented as $(1,0,1,0)$ in base two.

Basic components of IDEA being known, it is possible to describe the round function. IDEA handles blocks of 64 bits for plain and ciphertexts, and uses a master key of size 128 bits. The derivation algorithm produces at each round, from a given master key, subkeys of 96 bits. The block m_{i-1}, produced at the $(i-1)$th round, is used as input of the round function for the ith round. It is divided into four blocks, each of 16 bits, while the ith subkey is divided into six blocks of 16 bits, so that $m_{i-1} = m_{i-1}^1\, m_{i-1}^2\, m_{i-1}^3\, m_{i-1}^4$ and $k_i = k_i^1\, k_i^2\, k_i^3\, k_i^4\, k_i^5\, k_i^6$ where m_{i-1}^j and k_i^l are blocks of 16 bits for each $j = 1,2,3,4$ and $l = 1,2,3,4,5,6$. Notice that 16 satisfies the requirement that $2^{16}+1 = 65537$ is a prime number. As a consequence it is possible to use the three group laws previously introduced on blocks of 16 bits. The round function is based on a particular operation, denoted by MA, and called *multiplication-addition* or *MA-structure*, that takes four blocks x_1, x_2, y_1, y_2, each of 16 bits, in input and produces two blocks, $MA_1(x_1, x_2, y_1, y_2)$ and $MA_2(x_1, x_2, y_1, y_2)$, also 16 bits long. Mathematical relations between inputs and outputs of MA are the following:

$$\begin{array}{rcl} MA(x_1,x_2,y_1,y_2) & = & MA_1(x_1,x_2,y_1,y_2)\ MA_2(x_1,x_2,y_1,y_2) \\ MA_1(x_1,x_2,y_1,y_2) & = & MA_2(x_1,x_2,y_1,y_2) \boxplus (x_1 \odot y_1) \\ MA_2(x_1,x_2,y_1,y_2) & = & ((x_1 \odot y_1) \boxplus x_2) \odot y_2 \end{array}$$

where the second member of the first equality represents the concatenation of the blocks $MA_1(x_1, x_2, y_1, y_2)$ and $MA_2(x_1, x_2, y_1, y_2)$. The MA-structure is therefore composed of sophisticated and involved use of two of the three group operations, multiplication $\odot$ modulo $2^{16}+1$ and addition $\boxplus$ modulo 2^{16}. The MA-structure plays the same role as the function f from DES in fulfillment of confusion and diffusion, but unlike the latter, it is invertible whenever the inputs y_1 and y_2 are fixed. Indeed from the knowledge of the outputs $z_1 = MA_1(x_1, x_2, y_1, y_2)$, $z_2 = MA_2(x_1, x_2, y_1, y_2)$ of the MA-structure together with y_1, y_2, it is possible to recover x_1, x_2. The equation $z_1 = MA_2(x_1, x_2, y_1, y_2) \boxplus (x_1 \odot y_1)$ leads to the value of x_1. In fact, let us denote by a^{-1} (respectively $-a$) the inverse of a with respect to the operation $\odot$ (respectively $\boxplus$).

$$\begin{array}{rlcl} & z_1 & = & z_2 \boxplus (x_1 \odot y_1) \\ \Leftrightarrow & -z_2 \boxplus z_1 & = & x_1 \odot y_1 \\ \Leftrightarrow & (-z_2 \boxplus z_1) \odot y_1^{-1} & = & x_1. \end{array}$$

Then, injecting this value for x_1 into the equation $z_2 = ((x_1 \odot y_1) \boxplus x_2) \odot y_2$, is recovered the value of x_2. Indeed,

$$\begin{array}{rlcl} & z_2 & = & ((x_1 \odot y_1) \boxplus x_2) \odot y_2 \\ \Leftrightarrow & z_2 \odot y_2^{-1} & = & (x_1 \odot y_1) \boxplus x_2 \\ \Leftrightarrow & -(x_1 \odot y_1) \boxplus (z_2 \odot y_2^{-1}) & = & x_2 \\ \Leftrightarrow & -(((-z_2 \boxplus z_1) \odot y_1^{-1}) \odot y_1) \boxplus (z_2 \odot y_2^{-1}) & = & x_2. \end{array}$$

IDEA does not use any Feistel structure, invertible by construction, but we will see later the round function of IDEA to be also invertible. Surprisingly, invertibility of the MA-structure does not play any role in the deciphering process.

Let us precisely examine any round of IDEA. The ith round produces a block c_i of 64 bits divided into four blocks (16 bits each of them), which we denote by c_i^1, c_i^2, c_i^3 and c_i^4 such that $c_i = c_i^1\ c_i^2\ c_i^3\ c_i^4$. From a purely mathematical point of view, an IDEA round is given by the following formulae.

$$\begin{array}{rcl} c_i^1 & = & MA_2 \oplus (m_{i-1}^3 \boxplus k_i^3); \\ c_i^2 & = & MA_1 \oplus (m_{i-1}^4 \boxplus k_i^4); \\ c_i^3 & = & MA_2 \oplus (m_{i-1}^1 \odot k_i^1); \\ c_i^4 & = & MA_1 \oplus (m_{i-1}^2 \odot k_i^2) \end{array} \tag{12.1}$$

where we define

$$\begin{array}{rcl} MA_1 & = & MA_1((m_{i-1}^1 \odot k_i^1) \oplus (m_{i-1}^3 \boxplus k_i^3), (m_{i-1}^2 \odot k_i^2) \oplus (m_{i-1}^4 \boxplus k_i^4), k_5^i, k_6^i); \\ MA_2 & = & MA_2((m_{i-1}^1 \odot k_i^1) \oplus (m_{i-1}^3 \boxplus k_i^3), (m_{i-1}^2 \odot k_i^2) \oplus (m_{i-1}^4 \boxplus k_i^4), k_i^5, k_i^6). \end{array} \tag{12.2}$$

IDEA enciphering process is given as a sequence of 8 rounds for which the output c_i from the ith round is chosen as input for the following round. The ciphertext corresponding to the plaintext $m = m_0$ is not the block c_8, output of the eighth round. Indeed there is a final step: the ciphertext $c_9 = c_9^1\, c_9^2\, c_9^3\, c_9^4$ is computed by

$$\begin{array}{rcl} c_9^1 & = & c_8^1 \odot k_9^1; \\ c_9^2 & = & c_8^2 \odot k_9^2; \\ c_9^3 & = & c_8^3 \boxplus k_9^3; \\ c_9^4 & = & c_8^4 \boxplus k_9^4 \end{array} \tag{12.3}$$

where $k_9 = k_9^1\, k_9^2\, k_9^3\, k_9^4$ is a subkey of 64 bits (each of the k_9^i being composed of 16 bits), which also comes from the derivation algorithm applied to the master key.

Let us review the decryption process. First of all, let us explore how to recover the inputs m_{i-1}^j for $j = 1, \ldots, 4$ of round number i (for $1 \leq i \leq 8$) from the outputs c_i^j and subkeys k_i^l $(l = 1, \ldots, 6)$. Recall that the inverse of a block x under $\oplus$ operation is x itself. In particular, $x \oplus x$ is equal to the block with all bits equal to zero, neutral element for $\oplus$. From the definitions of c_i^j given by the equalities (12.1), the following result can be checked.

$$\begin{array}{rrcl} & c_i^1 \oplus c_i^3 & = & MA_2 \oplus (m_{i-1}^3 \boxplus k_i^3) \oplus MA_2 \oplus (m_{i-1}^1 \odot k_i^1) \\ \Leftrightarrow & c_i^1 \oplus c_i^3 & = & (m_{i-1}^3 \boxplus k_i^3) \oplus (m_{i-1}^1 \odot k_i^1). \end{array}$$

Similarly

$$\begin{array}{rrcl} & c_i^2 \oplus c_i^4 & = & MA_1 \oplus (m_{i-1}^4 \boxplus k_i^4) \oplus MA_1 \oplus (m_{i-1}^2 \odot k_i^2) \\ \Leftrightarrow & c_i^2 \oplus c_i^4 & = & (m_{i-1}^4 \boxplus k_i^4) \oplus (m_{i-1}^2 \odot k_i^2). \end{array}$$

Then notice that $(c_i^1 \oplus c_i^3)$ (respectively $c_i^2 \oplus c_i^4$) is the first (respectively the second) argument of the MA function according to the equalities (12.2). From this we see that under knowledge of all c_i^j and k_i^5, k_i^6, MA_1 and MA_2 can be computed. Finally using formulae (12.1) inputs from round number i, namely m_{i-1}^j, can be deduced since subkeys k_i^l are also known. For instance,

$$\begin{array}{rlcl} & c_i^1 & = & MA_2 \oplus (m_{i-1}^3 \boxplus k_i^3) \\ \Leftrightarrow & c_i^1 \oplus MA_2 & = & m_{i-1}^3 \boxplus k_i^3 \\ \Leftrightarrow & (c_i^1 \oplus MA_2) \boxplus (-k_i^3) & = & m_{i-1}^3. \end{array}$$

From c_9^j and subkeys $k_9^1, k_9^2, k_9^3, k_9^4$, $c_8 = c_8^1\, c_8^2\, c_8^3\, c_8^4$ is recovered: the equalities (12.3) are used. It can be easily shown that:

$$\begin{array}{rcl} c_9^1 \odot (k_9^1)^{-1} & = & c_8^1; \\ c_9^2 \odot (k_9^2)^{-1} & = & c_8^2; \\ c_9^3 \boxplus (-k_9^3) & = & c_8^3; \\ c_9^4 \boxplus (-k_9^4) & = & c_8^4. \end{array}$$

As previously claimed, invertibility of the MA-structure is not involved in the decryption process. In other terms, if f is any function that takes 4 blocs of 16 bits as input and produces two blocks of 16 bits as outputs, the encryption algorithm obtained, after substitution of the MA-structure by f in the IDEA algorithm, remains invertible and allows decryption process. So after all, what is the role of this function ? Actually diffusion requirement is based on MA. Indeed, each output subblock of MA depends on all input subblocks, in such a way that it ensures diffusion in a number of rounds less than DES.

Confusion is obtained by the mixed use of the three group operations, which, in a specific sense, are mutually incompatible, and thus allow to hide the algebraic structures used. As an example, the following properties can be listed. Let $\#$ and $\bigstar$ be two different operations from $\oplus$, $\boxplus$ and $\odot$ (for instance $\#$ is $\oplus$ and $\bigstar$ is $\boxplus$, or $\#$ is $\boxplus$ and $\bigstar$ is $\odot$).

1) No such pairs of operations $\#$, $\bigstar$ satisfy distributivity from one over the other, that is there are at least three blocks x, y, z, each of 16 bits, such that:

$$x\#(y\bigstar z) \neq (x\#y)\bigstar(x\#z);$$

2) No such pairs of operations $\#$, $\bigstar$ satisfy associativity, that is there are at least three blocks x, y, z of 16 bits such that:

$$x\#(y\bigstar z) \neq (x\#y)\bigstar z.$$

Nowadays: AES - Advanced Encryption Standard

On September 2, 1997, the NIST (National Institute of Standards and Technology) launched a call for proposals about a new cryptosystem to replace DES as a standard. The requirements were the following: a symmetric encryption algorithm, called AES (Advanced Encryption Standard), supporting blocks of size 128 bits, and keys of lengths 128, 192 and 256 bits. On August 20, 1998, the NIST announced the application of fifteen algorithms from twelve countries. A year later, after a detailed review of candidates, the NIST retained only five proposals, namely MARS, RC6, Rijndael, Serpent and Twofish. A second and last round was led by the NIST with help from the worldwide cryptographic community in order to select the winner. Dramatically no attacks were able to break any of the five last candidates [NEC 01]. Nevertheless other criteria, as algorithmic complexity or implementation characteristics, were applied to select, in year 2000, Joan Daemen and Vincent Rijmen's Rinjdael algorithm as new encryption standard AES. The official document [FIP 01], dated from November 26, 2001, approved AES as a cryptographic protection of sensitive electronic data (unclassified) of Federal agencies and departments of the U.S. government. In the same document are presented in detail full AES specifications.

AES supports 128, 192 or 256 bits long blocks as key formats, and plaintexts, ciphertexts have 128 bits. The choice of the length for keys depends on the level of protection needed by the communications (the longer they are, the more the security is enhanced); for instance, in a note from the American federal government [NSA 03], the National Security Agency (NSA) recommends to use keys of 192 or 256 bits for top-secret documents. Notice that the original Rinjdael was conceived to manipulate also other lengths for blocks and keys, but these ones were not retain for the final AES version. As its predecessor DES, AES operates on a certain number of rounds that depends on the length of the keys in a way described in the following table.

Keys	Number of rounds
128	10
192	12
256	14

Contrary to DES or IDEA, plaintexts and ciphertexts are not treated as blocks of bits, but as matrices of bytes, called *states*: the input of a round is a 4×4 matrix (4 rows, and 4 columns) of bytes entries. Thus a state is represented as the following matrix.

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$

where $a_{i,j}$ is a byte. Such a state represents the block

$$a_{0,0}\ a_{1,0}\ a_{2,0}\ a_{3,0}\ a_{0,1}\ a_{1,1}\ a_{2,1}\ a_{3,1}\ \dots\ a_{0,3}\ a_{1,3}\ a_{2,3}\ a_{3,3}.$$

Whatever the size chosen for the secret key, round subkeys are also represented by such 4×4 arrays of bytes (so they contain 128 bits).

In order to describe in detail the AES round function, some mathematical notions are required. We use the same notations as in the official document [DAE 99] available at the address `http://csrc.nist.gov/CryptoToolkit/aes/rijndael/Rijndael.pdf`.

AES uses operations that are defined on a finite field. A (commutative) *field* $\mathbb{K}$ is a set with at least two distinct elements, 0 and 1, and equipped with two internal composition laws, $+$ and $\times$ (to denote the second law, juxtaposition will also be used as the usual multiplication), such that

1) $\mathbb{K}$ with $+$ is a group with 0 for its neutral element (the inverse of x under this law will be denoted by $-x$, that is $x + (-x) = (-x) + x = 0$);

2) *Addition* $+$ is commutative: for every x, y in $\mathbb{K}$, $x + y = y + x$;

3) The set $\mathbb{K}^*$ of elements of $\mathbb{K}$ distinct from 0 is a group for *multiplication* $\times$ and its neutral element is 1 (the inverse of a non zero x in $\mathbb{K}$ for multiplication is denoted by x^{-1} and thus $xx^{-1} = x^{-1}x = 1$);

4) 0 is an absorbing element for multiplication: for every $x \in \mathbb{K}$, $x0 = 0x = 0$;

5) Multiplication is commutative: $xy = yx$ for every $x, y \in \mathbb{K}$;

6) Multiplication distributes over addition: $x(y + z) = xy + xz$ for all x, y, z in $\mathbb{K}$.

Among the class of all fields some have an infinite cardinality, while other, more interesting from an implementation point of view, have only finitely many elements. They are called the *finite fields*. A byte may be represented as an element of the finite field $\mathsf{GF}(2^8)$ with $2^8 = 256$ elements (" GF " stands for " Galois Field "). Thus a state may be seen as a 4×4 array with entries in this field.

AES round function has four basic components. Each of them is invertible which is an important difference compared to DES and IDEA. Indeed, AES round function is invertible as the composite of invertible maps while the corresponding property in DES or IDEA is not based upon invertible internal components. The first three components are independent of the round subkey, while the fourth is just entry-by-entry addition of bytes from the current state and those of the subkey. Each of these operations acts in a specific way on a state and promotes diffusion. A round thus consists in four stages as follows:

1) The function `ByteSub` (for *Byte Substitution*) is applied to a matrix A that represents the state at the input of the round. This map acts independently on each entry of A via an invertible transformation $\mathsf{S}_{\mathtt{RD}} : \mathsf{GF}(2^8) \to \mathsf{GF}(2^8)$. This function is essentially defined using

$$\mathsf{inv} : x \mapsto \begin{cases} 0 & \text{if } x = 0, \\ x^{-1} & \text{if } x \neq 0 \end{cases}$$

where x is a byte seen as en element of $\mathsf{GF}(2^8)$. More precisely $\mathsf{S}_{\mathsf{RD}} = \lambda \circ \mathsf{inv}$ where λ is an *affine* and invertible transformation of bytes; the term " affine " means that there exists a function α such that for all bytes x, y, $\alpha(x + y) = \alpha(x) + \alpha(y)$ (α is said to be *linear*), and a fixed byte β, interpreted as en element of $\mathsf{GF}(2^8)$, such that $\lambda(x) = \alpha(x) + \beta$. The linear map α also is invertible, and $\lambda^{-1}(x) = \alpha^{-1}(x - \beta)$.

Therefore the inverse of $\mathsf{S_{RD}}$ is obtained as $\mathsf{S_{RD}}^{-1} = \mathsf{inv} \circ \lambda^{-1}$ since, as easily checked, inv is its own inverse.

Graphically, ByteSub may be described as follows:

$$\texttt{ByteSub}\left[\begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix}\right] = \begin{pmatrix} b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\ b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} \end{pmatrix}$$

where each $b_{i,j} = \mathsf{S_{RD}}(a_{i,j})$.
Theoretical results assert that this function provides a very good level of confusion [DAE 02]; in particular, $\mathsf{S_{RD}}$, like DES S-boxes, is not linear since in general, $\mathsf{S_{RD}}(x + y) \neq \mathsf{S_{RD}}(x) + \mathsf{S_{RD}}(y)$. This first step " destroys " the group structure of $\mathsf{GF}(2^8)$ under addition. It is therefore an essential part in achieving the confusion within the cryptosystem. We also notice that ByteSub is invertible: in order to recover an input state A from $B = \mathsf{ByteSub}(A)$, it is sufficient to apply $\mathsf{S_{RD}}^{-1}$ on each entry of B; in other terms, the transformation obtained from ByteSub after substitution of $\mathsf{S_{RD}}$ by its inverse $\mathsf{S_{RD}}^{-1}$ is the inverse of ByteSub;

2) A shift, from left to right, is applied on the rows of the matrix $B = \mathsf{ByteSub}(A)$, output of ByteSub. These are cyclic shifts: for instance applying shift operation two consecutive times on the row

x_1	x_2	x_3	x_4

gives

x_3	x_4	x_1	x_2

This operation on the rows of the current state is called ShiftRow. The way the shift operates on a row depends on the row index. First row is not shifted, while the second is shifted one step to the right, the third two steps, and the fourth, three steps. Graphically ShiftRow acts on a matrix B with entries $b_{i,j}$ as follows.

$$\texttt{ShiftRow}\left[\begin{pmatrix} b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\ b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} \end{pmatrix}\right] = \begin{pmatrix} b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\ b_{1,1} & b_{1,2} & b_{1,3} & b_{1,0} \\ b_{2,2} & b_{2,3} & b_{2,0} & b_{2,1} \\ b_{3,3} & b_{3,0} & b_{3,1} & b_{3,2} \end{pmatrix}.$$

Like ByteSub, function ShiftRow is invertible. In order to recover the output matrix B of ByteSub it suffices to apply ShiftRow modified by the value of the applied shifts: the first row is not shifted, the second is now shifted three steps to the left, the third, two steps, and the fourth, only one step. This transformation aims at promoting diffusion in the cryptosystem;

3) The third step of the round, called MixColumn, operates at the column level of the current state. It is a matrix multiplication of each column of the state by the

same invertible 4×4 matrix M. Let $C = \mathsf{ShiftRow}(B)$ be the current state, result of ShiftRow, with entries denoted by $c_{i,j}$ ($i = 0, \ldots, 3$, $j = 0, \ldots, 3$). Let C_0, C_1, C_2, C_3 be the four columns of C, from left to right, in such a way C can be seen as concatenation, $[C_0 \mid C_1 \mid C_2 \mid C_3]$, of its columns. It follows that column number j (for $j = 0, \ldots, 3$) has the following form:

$$C_j = \begin{pmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{pmatrix}.$$

The multiplication of the column C_j by the matrix M (its entries, $m_{i,j}$, belong to the field $\mathsf{GF}(2^8)$) gives an other column $D_j = \begin{pmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{pmatrix}$. So $D_j = \mathsf{M}C_j$, and in matrix representation:

$$D_j = \mathsf{M}C_j \Leftrightarrow \begin{pmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{pmatrix} = \begin{pmatrix} m_{0,0} & m_{0,1} & m_{0,2} & m_{0,3} \\ m_{1,0} & m_{1,1} & m_{1,2} & m_{1,3} \\ m_{2,0} & m_{2,1} & m_{2,2} & m_{2,3} \\ m_{3,0} & m_{3,1} & m_{3,2} & m_{3,3} \end{pmatrix} \begin{pmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{pmatrix}.$$

Matrix multiplications lead to the following result concerning the values of $d_{i,j}$ for $i = 0, \ldots, 3$.

$$d_{i,j} = m_{i,0}c_{0,j} + m_{i,1}c_{1,j} + m_{i,2}c_{2,j} + m_{i,3}c_{3,j}.$$

The matrix multiplication of the columns of C produces four new columns D_0, D_1, D_2, D_3 that can be concatenated to build a 4×4 matrix D as $D = [D_0 \mid D_1 \mid D_2 \mid D_3]$. In short, MixColumn is formally defined as

$$D = \mathsf{MixColumn}(C) = [\mathsf{M}C_0 \mid \mathsf{M}C_1 \mid \mathsf{M}C_2 \mid \mathsf{M}C_3].$$

The fact that M is an invertible matrix is essential for the invertibility of MixColumn. The matrix M to be invertible means that there is another 4×4 matrix, denoted by M^{-1}, such that for every 1×4 matrix X (that is a matrix of one row and four columns, similar to a column C_j for instance)

$$\mathsf{M}^{-1}(\mathsf{M}X) = X. \tag{12.4}$$

In other terms, if M^{-1} is multiplied by the column that results from the product $\mathsf{M}X$, then X is obtained. Using this property we can prove MixColumn to be invertible. Indeed, let $\mathsf{MixColumn}^{-1}$ be the operation obtained from MixColumn after replacement of M by its inverse M^{-1}. Let us check that by applying $\mathsf{MixColumn}^{-1}$ to the state $D = \mathsf{MixColumn}(C)$, C is recovered.

$$\begin{aligned} \mathsf{MixColumn}^{-1}(D) &= [\mathsf{M}^{-1}D_0 \mid \mathsf{M}^{-1}D_1 \mid \mathsf{M}^{-1}D_2 \mid \mathsf{M}^{-1}D_3] \\ & \quad \text{(because } D = [D_0 \mid D_1 \mid D_2 \mid D_3]\text{)} \end{aligned}$$

$$= [\mathsf{M}^{-1}(\mathsf{M}C_0) \mid \mathsf{M}^{-1}(\mathsf{M}C_1) \mid \mathsf{M}^{-1}(\mathsf{M}C_2) \mid \mathsf{M}^{-1}(\mathsf{M}C_3)]$$
(since $D = \mathsf{MixColumn}(C)$)

$$= [C_0 \mid C_1 \mid C_2 \mid C_3]$$
(by property (12.4)).

4) The fourth and final step of an AES round is given by an addition of the MixColumn result D and the round subkey k, seen under the form of a 4×4 matrix of bytes. Therefore the following holds:

$$D + k = \begin{pmatrix} d_{0,0} & d_{0,1} & d_{0,2} & d_{0,3} \\ d_{1,0} & d_{1,1} & d_{1,2} & d_{1,3} \\ d_{2,0} & d_{2,1} & d_{2,2} & d_{2,3} \\ d_{3,0} & d_{3,1} & d_{3,2} & d_{3,3} \end{pmatrix} + \begin{pmatrix} k_{0,0} & k_{0,1} & k_{0,2} & k_{0,3} \\ k_{1,0} & k_{1,1} & k_{1,2} & k_{1,3} \\ k_{2,0} & k_{2,1} & k_{2,2} & k_{2,3} \\ k_{3,0} & k_{3,1} & k_{3,2} & k_{3,3} \end{pmatrix}$$

$$= \begin{pmatrix} d_{0,0}+k_{0,0} & d_{0,1}+k_{0,1} & d_{0,2}+k_{0,2} & d_{0,3}+k_{0,3} \\ d_{1,0}+k_{1,0} & d_{1,1}+k_{1,1} & d_{1,2}+k_{1,2} & d_{1,3}+k_{1,3} \\ d_{2,0}+k_{2,0} & d_{2,1}+k_{2,1} & d_{2,2}+k_{2,2} & d_{2,3}+k_{2,3} \\ d_{3,0}+k_{3,0} & d_{3,1}+k_{3,1} & d_{3,2}+k_{3,2} & d_{3,3}+k_{3,3} \end{pmatrix}.$$

So the result of $D + k$ is a 4×4 matrix E with byte entries $e_{i,j}$ given by

$$e_{i,j} = d_{i,j} + k_{i,j}.$$

This operation of round subkey addition is also invertible because

$$E = D + k \Leftrightarrow D = E + k$$

since addition in $\mathsf{GF}(2^8)$ is nothing else than a usual XOR, and thus $(D+k)+k = D$.

12.4. Prime Numbers and Public key Cryptography

This section is dedicated to the notion of public key cryptography (through one of its famous instance: the RSA algorithm) and its relation with arithmetic and, more precisely, prime numbers.

12.4.1. *Introduction*

Many cryptosystems use prime numbers. Most of them belong to the class of asymmetric encryption algorithms. In this section is briefly described one of the most famous, namely the RSA cryptosystem, and then the question of the construction of " cryptographic " prime numbers will be reviewed.

The very principle of public-key encryption was introduced by Diffie and Hellman [DIF 76] in 1976, but the authors were unable to provide an example of such algorithms. In asymmetric cryptography, two kinds of keys are involved: a *public key* and a *private key*. The first one is used for encryption, while the second enters the scene during the decryption process.

Let us review a communication between Alice and Bob, made confidential with a public-key algorithm. Let us assume that Alice wishes to send to Bob some confidential message. Alice finds Bob's public key k_B^{pub} which, as public, is known to everybody. Then, she encrypts her message, with this key, and sends it to Bob. Bob, who is the only one to know his private key k_B^{priv}, recovers the plaintext as a result of a decryption algorithm. Notice that for Bob to send a confidential message to Alice, he must use Alice's public key k_A^{pub} who will recover Bob's message with her own private key k_A^{priv}. In such a scenario, everybody has access to public keys, while private keys are kept secret.

Security of public-key encryption process is provided by infeasibility to solve some mathematical problem in a practical way, that is without requiring too much memory or time. In particular, the RSA algorithm is based on the problem of integer factorization or prime factorization of a number into its prime factors.

12.4.2. *The RSA Algorithm*

One of the first public-key algorithms, the RSA, was developed by Rivest, Shamir and Adleman [RIV 78].

Given two prime numbers p, q, it is easy to compute their product $n = pq$, even if both of them a very large. However it is difficult to *factorize* n, that is to recover its prime divisors when p, q are large enough (one thousand of decimal digits). The security of RSA is based on this assumption.

Keys Generation Step

Let p, q be two distinct prime numbers. Let $n = pq$ be the *RSA modulus*. An integer $e < (p-1)(q-1)$, coprime[5] to $(p-1)(q-1)$, is chosen. It is called the *encryption exponent*. Its inverse, d, modulo $(p-1)(q-1)$ is computed: d is the unique positive integer $x < (p-1)(q-1)$ such that

$$ex = a(p-1)(q-1) + 1 \tag{12.5}$$

5. Two integers a, b are coprime if, and only if, their greatest common divisor is 1.

for some integer a. The number d is called the *decryption exponent*. The ordered pair (n, e) is the public key, while (p, q, d) is the private key.

Enciphering Step

A plaintext m is a non-negative integer $< n$. Its corresponding ciphertext is the non-negative integer c, $0 \leq c < n$, given by

$$c = m^e \; mod \; n \; .$$

Notice that everybody is able to compute c from m, because n, e are given in the public-key.

Deciphering Step

In order to recover the plaintext m from the ciphertext c, computed as above, it is sufficient to compute

$$m = c^d \; mod \; n \; .$$

We notice that only a person with knowledge of d, part of the private key, is able to compute c^d.

Let us check the equality above to be true:

$$\begin{array}{rcl} c^d \; mod \; n & = & (m^d)^e \; mod \; n \\ & = & m^{ed} \; mod \; n \\ & = & mm^{a(p-1)(q-1)} \; mod \; n \text{ (according to (12.5))} \\ & = & m(m^{(p-1)(q-1)})^a \; mod \; n \\ & = & m \; mod \; n \\ & = & m \end{array}$$

since it can be checked that $m^{(p-1)(q-1)} = m \; mod \; n = m$.

12.4.3. *Primality and Pseudo-Primality*

Many asymmetric encryption schemes use prime numbers. It follows that the ways to test a number to be prime, or to construct prime numbers are very important issues in this context. The end of this chapter is devoted to a short presentation of some of these methods.

A problem must be solved in order to use a given prime number p in a public-key encryption: one must be sure that p is prime. This problem is obviously solved for small numbers, but it requires some techniques, called *primality tests*, for cryptographic relevant numbers (that is very large numbers).

In order to know if a given number is, or not, a prime number, it is possible to factorize this number. The Sieve of Eratosthenes is one of the oldest methods to achieve it. Nevertheless it becomes unusable when the number of decimal digits is large. The best factorization algorithms are able to compute the prime factors of numbers with an order of 200 decimal digits. But they are much lower than those needed in cryptography, and other criteria, than factorization, to determine an integer to be prime must be used.

Fermat's theorem states that if p is a prime number, then for every integer a, $1 < a < p - 1$, the following holds

$$a^{p-1} = 1 \; mod \; p.$$

Its reciprocal is false but may be used to develop weak and strong pseudo-primality tests [MEN 97]. Guaranteed primality tests have also been worked out. They are based on another reciprocal due to Lehmer and put into practice by Pocklington. Some of them will be reviewed. Nevertheless they are only used on pseudo-prime inputs.

12.4.4. *Pseudo-Primality Test*

The reciprocal of Fermat's theorem is false: for a non-prime integer n there is at least one integer a such that

$$a^{n-1} = 1 \; mod \; n.$$

This allows to define a base a weak pseudo-primality test for n.

1) Compute $a^{n-1} mod \; n$;

2) If the result is different from 1, then the integer n is a composite (it is not a prime number);

3) If it is equal to 1, then n is said to be a base a weak pseudo-prime.

Any prime number obviously is a pseudo-prime with respect to every base.

An algorithm, due to Strassen [SCH 71], computes $a^{n-1} mod \; n$ with a complexity $O(log_2 \; n)$, which is a multiplicative version of " Russian peasant " algorithm. It is an iterative program that connects the computation of a^n to that of $a^{n/2}$.

Strong Pseudo-Primality

Let n be an integer, pseudo-prime of base a. Let $n - 1 = d2^s$, with an odd d. If $a^d = 1 \; mod \; n$ or $a^{d2^r} = 1 \; mod \; n$ with $r < s$, then n is said to be *strong* *pseudo-prime of base a*. Experimentally, it is known that non-prime strong pseudo-prime are less numerous than non-prime weak pseudo-prime numbers. For instance, there are only 13 non-prime strong pseudo-prime numbers of bases 2, 3, 5 smaller than 25 10^9. Therefore, as primality test, strong pseudo-primality is better than weak pseudo-primality tests (for instance, for the same bases, they are an order of 2600 non-prime weak pseudo-prime smaller than 25 10^9).

12.4.5. *Guaranteed Primality Tests*

There is a theoretic primality test based on a reciprocal of Fermat's theorem conjectured and proved by Lehmer [LEH 35]. An implementation has been developed by Pocklington [POC 14].

Let N be an integer for which primality should be proved. Let $N-1 = R \times F$ be a partial factorization of $N-1$, F being a product of prime divisors of $N-1$, while R is not factorized. Let us assume that R and F are coprime, and $R < F$. Let $F = \prod_{j=1}^{n} q_j^{\beta_j}$ with prime numbers q_j. If there is some integer a such that $a^{(N-1)/q_j} - 1$ and N are coprime for every $j = 1, \cdots, n$, and $a^{N-1} = 1 \; mod \; N$, then N is a prime number.

The implementation is difficult to use because the choice of a is not deterministic. Nevertheless, only a partial number of prime factors of $N-1$ are needed to prove N to be prime (without any error).

Associated Construction Algorithm

It is possible to use primality test in order to construct prime numbers relevant for a cryptographic use. Let R be a prime integer, for instance obtained by a sieve method. Then smaller prime integers (one of them is 2) are chosen to constitute the part F. Using the previous algorithm, it is possible to find a new prime integer of length the double of that of R. Iterating this process, prime numbers p of arbitrary large value may be constructed with the property that $p-1$ has a large prime factor which is interesting to avoid factorization.

A method to obtain prime numbers is then the following.

1) Let $j = 1$;

2) Let p_j be a prime integer with 10 digits, obtained by a sieve method;

3) Compute a set of small prime numbers, and some of their powers, called a *base of primes*;

4) Pick at random in the base of primes some integers such that their product F_j is even, $F_j > p_j$;

5) Test weak pseudo-primality of $p_{j+1} = F_j.p_j + 1$;

6) If weak pseudo-primality does not hold, then change F_j and start again;

7) Change pseudo-primality base and check pseudo-primality with respect to this new base;

8) Iterate instruction 7 ten times;

9) Increment j. Iterate instructions 2 to 8 until a number p_j of expected size is obtained;

10) Test strong pseudo-primality of p_j. If it does not hold, then return to instruction 2;

11) Apply Pocklington's algorithm to p_j.

The integer p_j is a prime number with the expected number of digits.

12.5. Conclusion

In this chapter we focused on general principles of cryptography and on some of the famous encryption algorithms. It should be clear to everybody that an algorithm may be considered as sure only for a short period of time, and never in an absolute fashion. Indeed, cryptanalytic techniques are developed to break cryptosystems.

The evolution of mathematical technology to produce invertible functions relevant for a cryptographic use was highlighted in the second part of this chapter, where were presented in detail, and in a chronological order, DES, IDEA and AES. This evolution follows discovery of new mathematical objects (Feistel structures, more involved group structures, computations in finite fields, and so on). Therefore it is quite clear that new cipher algorithms will be designed in the future just as new mathematical objects will be discovered.

A new direction will perhaps be followed, namely *quantum cryptography* [BEN 84]. It is different from the kind of cryptography presented in this chapter, since it is based on principles of quantum mechanics rather than on mathematics. Security of those cryptosystems is ensured by the impossibility to duplicate an unknown wave function (Heisenberg uncertainty principle), or in other terms, impossibility to perform non-perturbative measures on a quantum system. Thus an adversary will measure some quantities (such as the spin of photons) in order to spy on confidential communications. Therefore the system will be perturbated so that Alice and Bob will be aware of the attack.

12.6. Bibliography

[BEN 84] BENNETT C., BRASSARD G., “Quantum cryptography: Public key distribution and coin tossing”, *Proceedings of the IEEE International Conference on Computers, Systems and Signal Processing*, p. 175–179, 1984.

[DAE 99] DAEMEN J., RIJMEN V., “AES proposal: Rijndael”, 1999.

[DAE 02] DAEMEN J., RIJMEN V., *The design of Rijndael: AES - the Advanced Encryption Standard*, Springer, 2002.

[DIF 76] DIFFIE W., HELLMAN M., “New directions in cryptography”, *Information Theory, IEEE Transactions on*, vol. 22, num. 6, p. 644–654, 1976.

[FEI 73] FEISTEL H., *Cryptography and computer privacy*, Scientific American, 1973.

[FIP 87] FIPS P., “81: DES Modes of Operation”, *National Bureau of Standards*, vol. 1, 1987.

[FIP 99] FIPS P., “46-3: Data Encryption Standard”, *National Institute for Standards and Technology*, vol. 25, 1999.

[FIP 01] FIPS P., “197: Advanced Encryption Standard”, *National Institute of Standards and Technology*, vol. 26, 2001.

[KER 83a] KERCKHOFFS A., “La cryptographie militaire (première partie)”, *Journal des Sciences Militaires*, vol. IX, p. 5–38, 1883.

[KER 83b] KERCKHOFFS A., “La cryptographie militaire (seconde partie)”, *Journal des Sciences Militaires*, vol. IX, p. 161–191, 1883.

[LAI 90] LAI X., MASSEY J., “A proposal for a new block encryption standard”, *Proc. EUROCRYPT*, vol. 90, p. 389–404, Springer, 1990.

[LAI 92] LAI X., MASSEY J., MURPHY S., “Markov ciphers and differential cryptanalysis”, *Advances in Cryptology - Eurocrypt*, vol. 91, p. 17–38, Springer, 1992.

[LEH 35] LEHMER D., “On Lucas’s test for the primality of Mersenne’s numbers”, *J. London Math. Soc.*, vol. 10, p. 162–165, 1935.

[MEN 97] MENEZES A., OORSCHOT P. V., VANSTONE S., *Handbook of applied cryptography*, CRC Press, 1997.

[NEC 01] NECHVATAL J., BARKER E., BASSHAM L., BURR W., DWORKIN M., FOTI J., ROBACK E., “Report on the development of the Advanced Encryption Standard (AES)”, *Journal of Research of the National Institute of Standards and Technology*, vol. 106, num. 3, p. 511–576, 2001.

[NSA 03] NSA, “15, Fact Sheet No. 1 National policy on the use of the Advanced Encryption Standard (AES) to protect national security systems and national security information. CNSS”, 2003.

[POC 14] POCKLINGTON H., “The determination of the prime or composite nature of large numbers by Fermat’s theorem”, *Proc. Cambridge Phil. Soc.*, vol. 18, p. 29–30, 1914.

[RIV 78] RIVEST R. L., SHAMIR A., ADLEMAN L., “A method for obtaining digital signatures”, *Commun. ACM*, vol. 21, p. 120–126, 1978.

[SCH 71] SCHONHAGE A., STRASSEN V., “Schnelle Multiplikation grosser Zahlen”, *Computing*, vol. 7, num. 3-4, p. 281–292, 1971.

[SHA 49] SHANNON C., *Communication theory and secrecy Systems*, Bell Telephone Laboratories, 1949.

[STI 06] STINSON D., *Cryptography: theory and practice*, Chapman and Hall/CRC, 2006.

[VER 26] VERNAM G., “Cipher printing telegraph systems for secret wire and radio telegraphic communications”, *Journal of the American Institute of Electrical Engineers*, vol. 45, p. 109–115, 1926.