Cryptography and Security
Lecture Notes

2024–25 Edition

Serge Vaudenay

EPFL
Lausanne, Switzerland
http://lasec.epfl.ch

Contents

Forewords

Cryptography is the science of information and communication security. It is used in pervasive applications such as credit cards, secure internet, smartphones, passports, wireless car keys or badges, etc. Cryptography provides a toolbox to build up security infrastructures. So far, using these tools correctly is hardly possible without fully understanding them. This is why mastering cryptography is often required in security jobs.

In these lecture notes, we study many cryptographic primitives. Some belong to what is called *symmetric cryptography* (in which a shared secret is assumed), such as symmetric encryption and message authentication. The former is used to make communication confidential. The latter is used to authenticate communication. Some other primitives belong to *public-key cryptography*, such as key agreement (to establish a secret between participants without assuming any pre-shared one), public-key cryptosystems (to protect confidentiality), and digital signature (to authenticate data).

These are the lectures notes for an annual course given since 1999. The course has evolved. For the version 2 of this course we published a textbook with Springer:

`http://www.vaudenay.ch/crypto/`

However, the course continued to evolve and we are now in its version 4.12. We collected all previous exams (with solutions) on the following page:

`http://lasec.epfl.ch/courses/exams_archives.shtml`

Cryptography requires backgrounds on a broad spectrum, especially in mathematics. We mostly have to deal with algebra and probabilities. Some reminders on algebra will be given.

What can be disturbing is that math may be formal or pretty dirty depending on the view point we adopt. This is inherent to cryptography. Sometimes, we design cryptographic schemes and we have to make formal proofs of security, even tough we cannot. For this, we formally prove security based on some hardness assumptions. However, sometimes, we try to break schemes and for that, it is often useless to deal formally with math. Any dirty mathematics may suffice as long as we show that a scheme is effectively broken. Cryptography is indeed about studying malicious behaviors and in this case, it is allowed to use math maliciously!

Finally, these lecture notes aim at introducing many different notions to give a taste of cryptography, but most of these notions are superficially covered. As for cryptanalysis, very little will be done. More details are part of a more advanced crypto class.

Chapter 1

Ancient Cryptography

This chapter focuses on encryption. It introduces elementary notions of cryptography, terminology, and important historical milestones. Basic principles such as the Kerckhoffs principle is stated. An approach of cryptography based on information theory is made. Essentially, we state what is the perfect notion of secrecy and what is the optimal way to achieve it.

Cryptography used to focus on enforcing the confidentiality of communication. During the "prehistory" of cryptography, it was used for military purposes, to protect business, or for private affairs. When it is used between two determined individuals, this can be done by establishing a secret convention to encode information. We refer to this as *security by obscurity*. This paradigm has some limitations, especially when one needs to communicate with more people, as it is cumbersome to invent a new convention each time. For that, some systems started to use the concept of an easy-to-configure *secret key*: people can use the same system, possibly developed by others, or even public, but select private keys on their own. The trend was to use more automatic procedures with the raise of the industrial era for communication. Finally, the *Kerckhoffs principle* was stated: the security should only rely on the secrecy of the key. (This will be discussed later.) The prehistory ended this way.

The *industrial era* was strongly influenced by the raise of the industrial communication (e.g., based on radio), and by the raise of industrial computing (namely: computers). Mass communication made the problem of selecting common algorithms more severe. Automatic computing made the security of cryptographic system more complicated since, suddenly, we could not bound the effort to break a system with the limit of humans, but could suddenly rely on new automatic devices which could work with virtually no limit.

Modern cryptography was marked by several results which appeared in different periods: the development of information theory and Shannon's result on cryptography, the appearing of public standards for cryptography such as DES, and the discovery of public-key cryptography in the 1970's.

1.1 Scope of Cryptography

There are a few technical terms which should be correctly used.

A *code* is a way to represent information. This notion makes no reference to cryptography. A *cipher* is a secret code. I.e., a secret way to represent information (so that it cannot be understood by unauthorized parties). In *coding theory*, people focus on the problem to keep the information available, e.g. by adding redundancy to protect against random noise. This is fundamentally different to cryptography, where we have to protect against a *malicious* process.

People often use *cryptology* as a broader notion than *cryptography*. They separate cryptography (designing cryptographic systems) from cryptanalysis (analyzing cryptographic systems), both notions being part of cryptology. Many people think of cryptanalysis as the action to break systems. This is however only one of the aspects of cryptanalysis. While analyzing, either we disprove security by breaking, but we can sometimes prove security as well.

Actually, modern cryptography does not focus solely on confidentiality. Many other cryptographic problems are considered: data integrity protection, data authentication, access control, timestamping, fair exchange, digital rights management, privacy, etc. These are used in many daily applications such as bank cards, e-commerce, mobile telephony, biometric passports, mobile communication, traceability in supply chains, pay-TV systems, car locks, public transport fees, electronic voting, etc.

In this lecture, we focus (but not only) on three fundamental problems which are encountered in the communication between a sender and a receiver:

- confidentiality (only the receiver shall receive the message),
- authentication (only the sender shall be able to send the message),
- integrity (the sent message shall match the received one).

We introduce the main cryptographic primitives which will be considered. First of all, we will discuss primitives which belong to the "symmetric cryptography" techniques. There is the *symmetric encryption*, which encrypt and decrypt messages with the same symmetric (secret) key. There is the *message authentication code*, which computes a tag for a message and verify the tag of a message using the same symmetric (secret) key. This is used to authenticate messages. We often include *hash functions* in this category of primitives. It is a deterministic function which computes a bitstring of fixed length for any message. The output is a kind of "fingerprint" of the message.

There is also the "public-key cryptography" technique. We consider *public-key cryptosystems* in which encryption and decryption are done with different keys, the decryption key being secret. What is new is that the encryption key can be made public without compromising privacy. We also consider *digital signatures* in which a secret key is used to "sign" a message and a public key is used to verify if a signature is valid. Usually, we also consider *key agreement protocols* which allow two participants to establish a common secret key over a public communication channel.

In the famous TLS standard which is mainly used to secure browsing on the internet, when a client (browser) wants to connect to a secure server, the server first sends its *certificate*, which is a signed document including the association between the address of the server (the URL) and a public key. The browser knows how to verify the certificate as the list of public keys of certificate authorities is already built in. This way, the browser can trust that a URL is associated to a given public key. The browser can then select a symmetric key and encrypt it using the public key of the server. The server is able to decrypt it and to retrieve the symmetric key. With this symmetric key, the client and the server can communicate securely using symmetric cryptography. We can see here that public-key cryptography is used to bootstrap secure communication, which is made using symmetric cryptography.

1.2 Cryptography Prehistory

Cryptography appeared at the same time as History: once people started to write, there was a need to protect information. In ancient civilizations such as ancient Egypt, the ability to write was the secret of scribes, transmitted from father to son. This was not public knowledge. Once the written language became widely spread, there was a need to transform the public encoding into a secret one.

Warriors in ancient Sparta used scytales. It was a common way to write messages by exchanging the position of some characters in the message. Typically, a message would be written in clear on a leather belt, when wrapped around a cylinder, and by writing along the axis of the cylinder. By unwrapping the belt, consecutive characters suddenly became distant on the belt, the distance matching the circumference of the cylinder. Moving characters is a technique known as *transposition*.

+	a	b	c	d	e	f	g	⋯
A	A	B	C	D	E	F	G	⋯
B	B	C	D	E	F	G	H	⋯
C	C	D	E	F	G	H	I	⋯
D	D	E	F	G	H	I	J	⋯
E	E	F	G	H	I	J	K	⋯
F	F	G	H	I	J	K	L	⋯
G	G	H	I	J	K	L	M	⋯
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	

Figure 1.1: Addition Table over the Alphabet

Caesar was rather using a convention to replace every character by another character, following a permutation of the entire alphabet. This is a technique known as *simple substitution*.

We could think that simple substitution is pretty secure as the set of all possible permutations of the alphabet is huge. In an alphabet of 26 characters (although the one of Caesar was a bit smaller), we have 26! possible permutations, i.e. roughly $2^{88.4}$. This means we can have a secret key of more than 88 bits, which is pretty long. Unfortunately, simple substitution can be broken by statistical analysis.

If we count on that the plaintext follows some biased distribution (typically, the plaintext would be written in Latin, and each character of the alphabet would be subject to a non-regular frequency), we can guess that the most frequent character of the ciphertext is the encryption of a very frequent character in the language of the plaintext. We can also analyze the frequency of consecutive characters. E.g., the frequency of *digrams* (which is a sequence of two characters) or even *trigrams* (sequences of three characters). Eventually, we can decrypt non-ambiguously. Anyway, the Caesar cipher used a permutation with a very special structure (namely, a circular rotation of the letters in the alphabet by two positions).

In the XVI-th Century appeared the *Vigenère cipher*. It was one of the first using a configurable secret key. A key would be a sequence of characters (e.g., "ABC"). To encrypt a plaintext, we would do an operation character-wise on the plaintext applied to the repetition of the key (i.e., "ABCABCABC…"). The operation between two characters, a plaintext character and a key character, consists of applying a circular rotation of the alphabet which maps "a" to the key character. For instance, if the key character is "B", we rotate the alphabet by one position to map "a" to "B". So, if the plaintext character is "h" and the key character is "B", the ciphertext character is "I". If we denote this operation with a +, it is defined by the addition table on Fig. 1.1. We can prove that the alphabet together with this + rule forms an *Abelian group* and that it is *isomorphic* to $\mathbf{Z}_{26}$, as it will be seen in the next chapter.

If we write the message in a table with a number of columns corresponding to the length of the key, we observe that all characters in the same column are mapped through the same alphabet rotation. So, we could do some statistical analysis in each column separately.

An interesting observation was made by Kasiski in the XIX-th Century. If we find a frequent pattern of consecutive characters in the ciphertext, it most likely corresponds to the encryption of a frequent pattern in the plaintext and that the distance between the occurrences are multiple of the key length. This could be used to deduce the key length from the ciphertext. It is known as the *Kasiski test*.

Another common tool to analyze the Vigenère cipher is the *index of coincidence*. Given a sequence of characters $x_1, \ldots, x_n$, the index of coincidence is the probability that $x_I = x_J$, given I and J uniformly distributed among pairs such that $1 \leq I < J \leq n$. If Z denotes the alphabet and n_c is the number of occurrences of a character $c \in Z$ in the sequence, we have

$$\mathsf{Index}(x_1, \ldots, x_n) = \Pr_{I<J}[x_I = x_J] = \frac{1}{n(n-1)} \sum_{\substack{1 \leq i,j \leq n \\ i \neq j}} 1_{x_i = x_j} = \sum_{c \in Z} \frac{n_c(n_c - 1)}{n(n-1)}$$

The index of coincidence is invariant under simple substitution: for any permutation σ of Z, we have

$$\mathsf{Index}(\sigma(x_1), \ldots, \sigma(x_n)) = \mathsf{Index}(x_1, \ldots, x_n)$$

Similarly, the index of coincidence is invariant under transposition: for any permutation σ of $\{1, \ldots, n\}$, we have

$$\mathsf{Index}(x_{\sigma(1)}, \ldots, x_{\sigma(n)}) = \mathsf{Index}(x_1, \ldots, x_n)$$

We can compute the expected value of the index of coincidence by

$$\begin{aligned} E(\mathsf{Index}(x_1, \ldots, x_n)) &= \frac{1}{n(n-1)} \sum_{\substack{1 \le i,j \le n \\ i \ne j}} \Pr[x_i = x_j] \\ &= \sum_{c \in Z} f_c^2 \end{aligned}$$

By assuming that the x_i's are independent and identically distributed, with $\Pr[x_i = c] = f_c$. The expected index of coincidence of a sequence of uniformly distributed characters is thus 0.039 (by taking $f_c = \frac{1}{|Z|}$ and $|Z| = 26$) while it is close to 0.065 for an English text (using the frequency table of characters in English). As n goes to infinity, the real index of coincidence become close to the expected one. Hence, we can find the length of the Vigenère key by making a guess and checking that the index of coincidence of every column is high. Then, we can recover the alphabet rotation of each column.

1.3 Pre-Modern Industrial Cryptography

In 1918, Siemens invented Enigma: an electro-mechanical encryption device which could be used to type a message and plugged to a radio transmitter. This device was patented (so, it is public), and easily configurable by setting up an initial state.

This device was massively used by the Germans during the Second World War. The communications were also massively intercepted. To decrypt it, several mathematicians joined their forces, and people (including Alan Turing) had to invent computers to make the decryption task automatic. So, computer science originally appeared to solve a cryptanalysis task.

Without entering into the technical design of the Enigma device, we can define one instance of the Enigma cipher as follows. First of all, the system assumes a common (public) permutation π of the 26-letter alphabet and a set of (say five) permutations S. The permutation π is such that it is an involution (i.e., we have $\pi(\pi(x)) = x$ for all letter x) which has no fixed point (i.e., we have $\pi(x) \neq x$ for all letter x). We call *reflector* the permutation π. Permutations in the set S are called *rotors*. We further define ρ, the circular rotation of the alphabet. E.g., $\rho(\text{a}) = \text{b}$, $\rho(\text{b}) = \text{c}$, ... Given a rotor $\alpha \in S$ and an integer $i \in \mathbf{Z}$, we say that α in position i defines the permutation $\alpha_i = \rho^i \circ \alpha \circ \rho^{-i}$. (Technically, the rotor is a wheel connecting 26 input plugs to 26 output plugs, and when the rotor rotates by $\frac{1}{26}$th of a complete circle, it is equivalent as making the input rotate by ρ^{-1} and the output rotate by ρ at the same time. So, α_i is the permutation defined after i rotations.)

In Enigma, there is a *plug board* of 26 plugs and we can connect plugs by a cable. Concretely, we have 6 cables and we can select 6 non-overlapping pairs of distinct plugs. After connection, this defines an additional permutation σ which lets fixed letters corresponding to an unconnected plug and permutes the other letters by pairs. I.e., the set of all $\{x; \sigma(x) = x\}$ has cardinality 14 and σ is an involution: for all x, we have $\sigma(\sigma(x)) = x$. A key of Enigma is a tuple consisting of

- an order triplet (α, β, γ) of different rotors, i.e., $\alpha, \beta, \gamma \in S$;
- an integer a;
- an involution σ with 14 fixed points.

We can compute the total number of keys. The total number of selection of rotors is $5 \times 4 \times 3 = 60$ if S has only 5 rotors. The number of possibilities for a is $26^3 = 17\ 576$. The number of possible σ is

$$\binom{26}{14} \times 11 \times 9 \times 7 \times \cdots \times 1 = 100\ 391\ 791\ 500$$

Finally, we have

$$60 \times 17\ 576 \times 100\ 391\ 791\ 500 \approx 2^{57}$$

so an Enigma key is equivalent to a 57-bit key.

The plaintext x is a sequence of letters $x_1, \ldots, x_m$. It is encrypted on-the-fly into $y = (y_1, \ldots, y_m)$. To encrypt x_i, we write $i - 1 + a$ in basis 26 and obtain $i - 1 + a = i_3 i_2 i_1$. I.e., $i - 1 + a = 26^2 i_3 + 26^2 i_2 + i_1$ and $i_1, i_2, i_3 \in \{0, 1, \ldots, 25\}$. Then, we define

$$y_i = \sigma^{-1} \circ \alpha_{i_1}^{-1} \circ \beta_{i_2}^{-1} \circ \gamma_{i_3}^{-1} \circ \pi \circ \gamma_{i_3} \circ \beta_{i_2} \circ \alpha_{i_1} \circ \sigma(x_i)$$

In the Enigma device, typing x_i sends a signal to the corresponding plug, goes through the involution σ, then through the three rotors α, β, γ, then through the reflector, then comes back through the rotors γ, β, and α, the permutation σ again, and lights up a lamp corresponding to y_i. Every time we type a key, the rotor α moves by one position (i.e., the least significant i_1 is incremented). Every 26 keystrokes, the rotor β moves by one position (i.e., i_2 is incremented). Every 26^2 keystrokes, the rotor γ moves by one position (i.e., i_3 is incremented). Also: the initial position of the rotors corresponds to a.

Interestingly, Enigma is also an involution: by encrypting y_i in the very same position of the rotors, we must obtain x_i. This means that the encryption and the decryption operation are exactly the same.

Laws in modern cryptography. In modern cryptography, we must keep in mind some fundamental laws.

Firstly, we should not make the security of a system dependent on the secrecy of its design. Based on that cryptographic systems are not designed by their users, or that they may be stolen, we cannot assume that the adversary ignores the design of the system. This law is known as the *Kerckhoffs principle*. Many people misread this and claim it says that cryptographic systems must be publicly known. This is a common mistake.

Secondly, in a network of n users, the number of potential pairs of users willing to communicate securely has the order of magnitude of n^2. So, we should think of a common system and keep it configurable by an easy-to-select secret key.

Thirdly, we must keep in mind that the computational power of available devices is always increasing. Moore's law says that the growth is exponential. As the computational power doubles every cycle (say 18 months), breaking a key by exhaustive search is twice faster after a cycle. Considering that a standard computer CPU could try about one million keys per second in the 2007 technology, breaking a 128-bit key by exhaustive search within 14 billion years requires 770 000 such computers. Assuming the Moore law goes on until 2215, a single computer would do the same job within a second! This leads us to an interesting philosophical answer: if we have 14 billions years to spend to break a 128-bit key, it is better to launch the Big Bang, take vacations, wait until computers invent by themselves, then go down to Earth in 2215 to buy one, rather than make too many machines work for too long.

1.4 Cryptography and Information Theory

We consider the set $\{0, 1\}^n$ of bitstrings of length n with the bitwise XOR operation, i.e.

$$(x_1, \ldots, x_n) \oplus (y_1, \ldots, y_n) = (x_1 \oplus y_1, \ldots, x_n \oplus y_n)$$

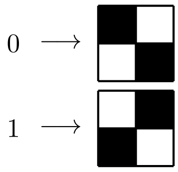

Figure 1.2: Pixel Encoding

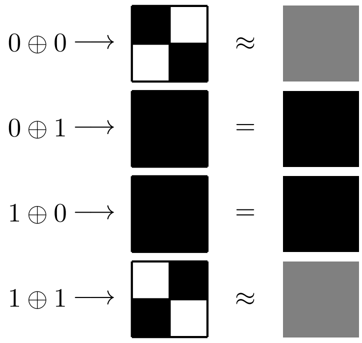

Figure 1.3: Visual Pixel Overlay Decoding

with $a \oplus b = (a+b) \bmod 2$ for $a, b \in \{0, 1\}$. Note that $\oplus$ is associative, commutative, $(0, \ldots, 0)$ is neutral, and each element is self-inverse since

$$x \oplus x = (0, \ldots, 0)$$

So, we have an Abelian group in which all elements are self-inverse. We define an encryption scheme over this group. Given a message X in this set and a key K in the same set, the encryption of X is $Y = X \oplus K$. To decrypt, we just compute

$$Y \oplus K = (X \oplus K) \oplus K = X \oplus (K \oplus K) = X \oplus (0, \ldots, 0) = X$$

The *Vernam cipher* [83] assumes that K is of same length as X, uniformly distributed, and used to encrypt only once. We often call it *one-time-pad*.

The Vernam cipher can be nicely implemented without any computing devices as observed by Naor and Shamir [60]. This is called *visual cryptography*. The idea is to decrypt images by using transparent paper, eyes, and brain! Both the ciphertext and the key are black-and-white images consisting of pixels. As shown on Fig. 1.2, each pixel is encoded by a pattern which is half white and half black. The encoding of the pixel 0 is the complement of the encoding of the pixel 1. So, an image is encoded by tiles which encode each pixel. If we overlay the ciphertext and the key and look through the transparent papers, a pixel b which is put on another pixel b will look like the encoding of the pixel b, i.e., it will be half white and half black. This is what the eye can see but from far away, the brain will take it as a grey tile. (See Fig. 1.3.) Interestingly, $b \oplus b = 0$. This is the corresponding bit of the decryption operation. We say that a grey tile corresponds to a 0 bit in the plaintext. If we overlay a pixel 0 and a pixel 1, as they are complement of each other, the eye and brain will see a completely black tile. Since $0 \oplus 1 = 1$, we say that a black tile corresponds to a 1 bit of the plaintext. So, the brain will interpret the grey and black tiles and will "see" the plaintext. As an application, someone can just hold the key as printed on a transparent paper and use it to decrypt ciphertexts which could be sent by fax.

We must not encrypt twice with the same key. Otherwise, the system would be insecure. Indeed, assume that Y_1 resp. Y_2 is the encryption of X_1 resp. X_2 under the same key K. We have

$$Y_1 \oplus Y_2 = (X_1 \oplus K) \oplus (X_2 \oplus K) = X_1 \oplus X_2$$

So, if the plaintexts X_1 and X_2 have some structure, their XOR may leak some information and this XOR can be computed from the ciphertexts only. As an example, if X_1 and X_2 are black and white digital images with a lot of blank, we can see in $Y_1 \oplus Y_2$ the blank zones from both X_1 and X_2 and the shape of the two pictures. So, the plaintexts leak.

Similarly, we must use a key which is at least as long as the plaintext. If it is too short, some part of the plaintext may remain in clear.

Finally, the key must be uniformly distributed. Otherwise, $Y = X \oplus K$ will follow a biased distribution which may leak some information about X. We will see that Y is uniformly distributed otherwise.

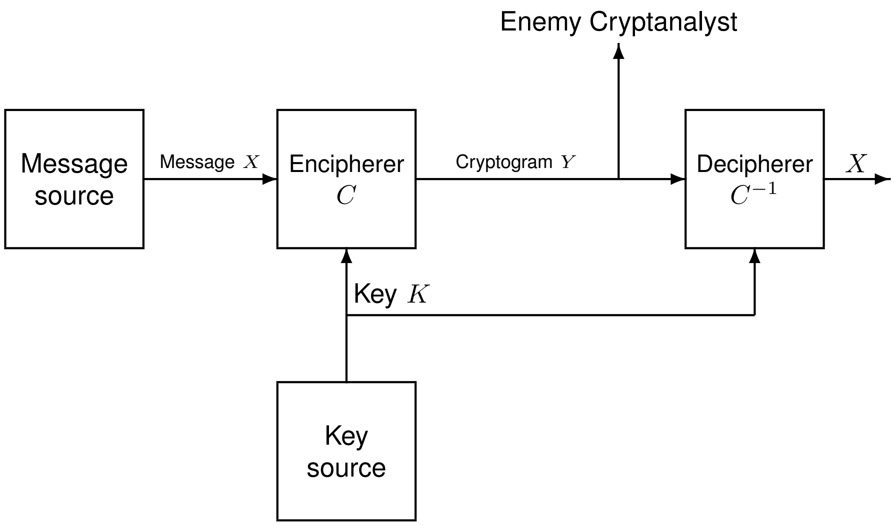

Figure 1.4: The Shannon Encryption Model

It is pretty cumbersome to require a key as long as the plaintext every time we need to encrypt. It is unsuitable for most of applications. In the case of the red telephone, during the Cold War, this was making sense: USSR and USA could exchange some long keys K over a secure slow channel (typically: an ambassador with a case full of keys flying from one capital to the other) in order to prepare emergency calls over a fast insecure channel (typically: radio).

When Vernam published this cipher in 1926, it was believed to be perfectly secure but there were no available formalism to express it or prove it. We had to wait until 1949 to obtain the appropriate formalism by Shannon.

In the Shannon model [77], there are two algorithms C and C^{-1} to encrypt and to decrypt, respectively. (See Fig. 1.4.) The plaintext X and the key K are two independent random variables. The distribution of K is specified by the encryption scheme, and the one of X is dependent on the application. We stress that in the Shannon model, the distribution of X is part of the cryptographic system. This is not so common now as we favor systems working for arbitrary plaintext distributions. The ciphertext $Y = C_K(X)$ is also a random variable, with a distribution induced by the distributions of X and K. The cryptographic scheme is *correct* if $\Pr[C_K^{-1}(C_K(X)) = X] = 1$.

As an example, we can formalize a generalized form of the Vernam cipher in this model. Given a finite group G (with additive notations), imagine that the plaintext message X follows a given distribution D in G. We use a key K which is independent and uniformly distributed in G. We define $Y = K + X$ to encrypt and $X = (-K) + Y$ to decrypt. This cipher is to be used for a single X. (More encryptions need to set up new keys.)

The adversary has access to Y only and tries to retrieve some information about X. We say that the cryptographic scheme provides *perfect secrecy* if the value y taken by Y leaks no *a posteriori* information on X which was not already known *a priori*. This is formalized by $\Pr[X = x|Y = y] = \Pr[X = x]$ for all x and all y such that $\Pr[Y = y] \neq 0$. Actually, perfect secrecy is defined by the following result.

Definition 1.1 (Perfect Secrecy). *Given some independent random variables X and K in a discrete space and a cipher C in the Shannon model to define $Y = C_K(X)$, the following properties are equivalent:*

- *for all x and y such that* $\Pr[Y = y] \neq 0$*, we have* $\Pr[X = x|Y = y] = \Pr[X = x]$*;*
- *X and Y are independent;*

- *$H(X|Y) = H(X)$, where H denotes the Shannon entropy.*[1]

If these properties are satisfied, we say that the cipher provides perfect secrecy*.*

Proof. The equivalence of the first two properties comes from the definition of independence. By definition of the conditional entropy, we have $H(X|Y) = H(X,Y) - H(Y)$. So, the equivalence with the last property is a consequence of the result saying that X and Y are independent if and only if $H(X,Y) = H(X) + H(Y)$. □

The above notion of perfect secrecy is relative to a specific distribution on the plaintext X and we can wonder what is the influence of the choice of this distribution on perfect secrecy. The following result shows that if we have perfect secrecy for a plaintext distribution giving a nonzero probability to any possible plaintext in the domain of C, then we have perfect secrecy for all distributions on this domain.

Theorem 1.2. *Let C_K be a cipher with K following a given distribution and input plaintext on a given domain $\mathcal{X}$. Let p and p' be two distributions on $\mathcal{X}$ such that the support of p is $\mathcal{X}$. In what follows, we denote by $\Pr_p$ resp. $\Pr_{p'}$ the probability using the distribution p resp. p'.*

C_K provides perfect secrecy with p implies that C_K provides perfect secrecy with p'.

Proof. Let $Y = C_K(X)$ be the ciphertext. Let y be such that $\Pr_{p'}[Y = y] \neq 0$. (Note that $\Pr_{p'}[Y = y]$ is the probability that $Y = y$ given the distribution p' on X.) We need to prove that for any x, $\Pr_{p'}[Y = y] = \Pr_{p'}[Y = y|X = x]$.

Since $\Pr_{p'}[Y = y] \neq 0$, there exist k and x_0 such that $C_k(x_0) = y$, $\Pr[K = k] \neq 0$, and $p'(x_0) \neq 0$. Thanks to p having a full support, we have $p(x_0) \neq 0$ so $\Pr_p[Y = y] \neq 0$.

Let us take an arbitrary x. Due to perfect secrecy, we have $\Pr_p[Y = y] = \Pr_p[Y = y|X = x]$. But $\Pr_p[Y = y|X = x] = \Pr[C_K(x) = y]$ does not depend on the distribution of X so we have $\Pr_p[Y = y|X = x] = \Pr[C_K(x) = y] = \Pr_{p'}[Y = y|X = x]$. Hence, $\Pr_p[Y = y] = \Pr_{p'}[Y = y|X = x]$.

Then

$$\begin{aligned}
\Pr_{p'}[Y = y] &= \sum_x \Pr_{p'}[Y = y|X = x]p'(x) \\
&= \sum_x \Pr_p[Y = y]p'(x) \\
&= \Pr_p[Y = y]\sum_x p'(x) = \Pr_p[Y = y] \\
&= \Pr_{p'}[Y = y|X = x]
\end{aligned}$$

and we have proven perfect secrecy with p'. □

Given this formalism, it is straightforward to prove that for any distribution of X, the Vernam cipher provides perfect secrecy. Actually, the good distribution of Y comes from the following lemma:

Lemma 1.3. *Let X and K be two independent random variables over a discrete group. We assume that K is uniformly distributed. Then, $Y = K + X$ is also uniformly distributed, and independent from X.*

Proof. Let x and y be two group elements. We have $\Pr[X = x, Y = y] = \Pr[X = x, K = y - x] = \frac{1}{\#G}\Pr[X = x]$, where $\#G$ is the cardinality of the group. By summing over all x, we obtain $\Pr[Y = y] = \frac{1}{\#G}$. So, $\Pr[X = x, Y = y] = \Pr[X = x]\Pr[Y = y]$: X and Y are independent. □

Since the Vernam cipher imposes that the key is as long as the plaintext and that it provides perfect secrecy, a natural question consists of wondering if there are ciphers with perfect secrecy without this drawback. Shannon answered by the negative.

[1]We provide some reminders on the Shannon entropy in Appendix B.

Theorem 1.4 (Shannon 1949). *In a correct cryptographic system providing perfect secrecy, we have* $H(K) \geq H(X)$.

Proof. Once the value of K is determined, we can compute $Y = C_K(X)$ or $X = C_K^{-1}(Y)$ due to the correctness of the cryptographic system. So, we can make changes in the variable of the sum defining the conditional entropy and obtain $H(Y|K) = H(X|K)$. Since X and K are independent, we have $H(X|K) = H(X)$. Thus, we obtain $H(Y|K) = H(X)$.

Due to Lemma B.2 in Appendix, we have $H(Y) \geq H(Y|K)$. We thus have $H(Y) \geq H(X)$. Note that we did not use perfect secrecy. So, $H(Y) \geq H(X)$ is a general property of correct cryptographic systems.

If the value of X is determined, we can compute $Y = C_K(X)$ from K. So, many terms in the sum of the joint entropy of $K|X$ and $Y|X$ vanish and we obtain $H(Y, K|X) = H(K|X)$. Since X and K are independent, we have $H(K|X) = H(K)$. So, $H(Y, K|X) = H(K)$.

Due to Lemma B.2 in Appendix, we have $H(Y, K|X) \geq H(Y|X)$. Thus, $H(K) \geq H(Y|X)$. Again, this is a general property of correct cryptographic systems.

Now, if we have perfect secrecy, we have $H(Y|X) = H(X|Y) + H(Y) - H(X) = H(Y)$. So, we have $H(K) \geq H(Y) \geq H(X)$. □

There is another form of the Shannon Theorem which is more common.

Theorem 1.5 (Shannon 1949). *In a correct cryptographic system providing perfect secrecy, we have*

$$\#\{k; \Pr[K = k] \neq 0\} \geq \#\{x; \Pr[X = x] \neq 0\}$$

So, we cannot have less keys than messages.

Proof. Let y be a fixed value such that $\Pr[Y = y] \neq 0$. Since X and K are independent, we have

$$\Pr[X = x, Y = y] = \Pr[X = x, C_K(x) = y] = \Pr[X = x] \Pr[C_K(x) = y]$$

For all x such that $\Pr[X = x] \neq 0$, perfect secrecy implies $\Pr[C_K(x) = y] = \Pr[Y = y|X = x] = \Pr[Y = y] \neq 0$. Consequently, for all x such that $\Pr[X = x] \neq 0$, there must exist one k such that $C_k(x) = y$ and $\Pr[K = k] \neq 0$. Let us write it $k = f(x)$.

Due to correctness, for any x which has a nonzero probability, we have $C_{f(x)}^{-1}(y) = x$. So, $f(x) = f(x')$ implies $x = x'$. Consequently, we have an injection from the set of all possible x to the set of all possible k. We deduce the inequality. □

Interestingly, we could also prove that the set of possible X cannot be infinite and enumerable.

Theorem 1.6. *In a correct cryptographic system providing perfect secrecy,* $\{x; \Pr[X = x] \neq 0\}$ *cannot be infinite and enumerable.*

I.e., we cannot have perfect secrecy over an infinite enumerable message space.

Proof. We defined cryptographic systems for discrete sets. So, they are enumerable. We have to show that $\{x; \Pr[X = x] \neq 0\}$ must be a finite set.

Let y be a fixed value such that $p = \Pr[Y = y] \neq 0$. (Such y must exist because we have an enumerable domain.) Due to perfect secrecy, we have $p = \Pr[Y = y] = \Pr[C_K(x) = y]$ for all possible x. Due to correctness, $C_K(x) = y$ implies $C_K^{-1}(y) = x$. So, $\Pr[C_K^{-1}(y) = x] \geq \Pr[C_K(x) = y] = p$ for all possible x. Furthermore, we know that by summing $\Pr[C_K^{-1}(y) = x]$ over all possible x, we must obtain 1. So,

$$1 \geq \sum_{x; \Pr[X=x] \neq 0} \Pr[C_K^{-1}(y) = x] \geq p.\#\{x; \Pr[X = x] \neq 0\}$$

Consequently, $\#\{x; \Pr[X = x] \neq 0\} \leq \frac{1}{p}$. So, it is a finite set. □

For practical reasons, secure ciphers over a domain of finite bitstrings also leak the length of a message. Indeed, we ideally want that the encryption of a short message (e.g. 100 bytes) remains short, while we also want to encrypt messages of several megabytes. Hence, the length often leaks and there is little we can do about it: we must live with it.

So now, we conclude that there exists a cipher providing perfect secrecy. It has a drawback of using keys as long as the message to encrypt, but we can show that it is necessary for perfect secrecy. Furthermore, it may be the simplest cipher we could imagine, given this drawback. So, we have modeled encryption, security, and identified the best possible cipher. Could we stop the lecture here? Certainly not, because there is an important notion which is not captured at all in this theory: *complexity*.

So far, the Shannon notion for perfect secrecy wonders if recovering the information *could* be possible, no matter the cost. As computer science began, it became clear that feasibility in theory does not mean feasibility in practice. So, we should add the notion of *cost* — that we usually call complexity — of the information recovery in the security model. Ideally, we would like to say that recovering some information about the confidential message is too expensive for the adversary so that the system is secure. This way, we could hope to have secure ciphers better than the Vernam cipher in terms of efficiency with reusable and short key.

Chapter 2

Diffie-Hellman Cryptography

This chapter recalls algebraic notions to introduce the Diffie-Hellman key exchange protocol. For that, basic notions of group theory are recalled, as well as algorithmic methods.

2.1 Arithmetics and $\mathbf{Z}_n$

A *prime number* is a positive integer having exactly two positive factors: 1 and itself. Integers have a unique factorization, up to permutation, into a product of prime numbers and a unit which is either $+1$ or -1.

For all $a \in \mathbf{Z}$ and $n > 0$, there exists a unique way to write $a = qn + r$ with $q, r \in \mathbf{Z}$ and $0 \leq r < n$. This is the *Euclidean division*. We call r the *remainder* and we write it $r = a \bmod n$.

We stress that there exist two different notations for mod: one (the above one) is a dyadic operation, and the other is an attribute to an equivalence. Indeed, we write $a \equiv b \pmod{n}$ to denote that $b - a$ is divisible by n, or equivalently, that $a \bmod n = b \bmod n$: somehow, a and b are "equal", but only modulo n.

The modulo operation is the basis of the definition for $\mathbf{Z}_n$. We can define it simply by saying that it is the set $\{0, 1, \ldots, n-1\}$ together with new arithmetic operations $\boxplus$ and $\boxtimes$. We define

$$a \boxplus b = (a + b) \bmod n \qquad a \boxtimes b = (a \times b) \bmod n$$

Then, we easily show that they are associative and commutative, that 0 is neutral for $\boxplus$, that 1 is neutral for $\boxtimes$, and that every element has an inverse for $\boxplus$, but not necessarily for $\boxtimes$. We have distributivity of $\boxtimes$ over $\boxplus$ as well.

2.2 Some Notions of Group Theory

Definition 2.1. *A* group G *is a set with a two-input operation* $+$ *such that*

- *the set is* closed *under the operation: for all a and b in G, we have $a + b \in G$;*
- *the operation is* associative*: for all $a, b, c \in G$, we have $a + (b + c) = (a + b) + c$;*
- *there exists a* neutral element 0 *(which is necessarily unique): for all $a \in G$, we have $a + 0 = 0 + a = a$;*
- *every set element has an* inverse*: for all $a \in G$, there exists an element in G denoted by $-a$ such that $a + (-a) = (-a) + a = 0$.*

We say the group is Abelian *if the operation is further commutative: for all a and b in G, we have $a + b = b + a$.*

As we have a single operation in groups, there exist essentially two families of notations: the additive and the multiplicative notation. With additive notations (like in the above definition), the operation is denoted by $+$, $\oplus$, $\boxplus$ or any similar symbol, the neutral element is denoted 0, $\mathcal{O}$, ..., and the inverse of x is denoted $-x$. With multiplicative notations, the operation is denoted $\times$, $\otimes$, $\boxtimes$, $\cdot$, the neutral element is 1, I, e, and the inverse of x is $\frac{1}{x}$, x^{-1}, ... Due to associativity, we can add (resp. multiply) an element x to itself n times and define $n.x$ (resp. x^n).

As usual in algebra, mappings from a group to another group which preserve the group operation equalities are called *group homomorphisms*. They are *isomorphisms* if they are further bijective. An isomorphism can be seen as a change of notations as it preserves the algebraic structures. For instance, the addition tables on Fig. 1.1, p. 3, gives to the set $G = \{\mathtt{A}, \mathtt{B}, \ldots, \mathtt{Z}\}$ a group structure which is isomorphic to $\mathbf{Z}_{26}$ by $f(\mathtt{A}) = 0$, $f(\mathtt{B}) = 1$, etc.

We can consider several group constructors:

- we can consider *subgroups* of a bigger group, e.g. the smallest group containing a given subset of elements, which is then called the subgroup generated by these elements;
- the *product* of two groups, and the group raised to a power n, the group of functions from a given set to a group;
- the *quotient* of an Abelian group by one of its subgroup.

An important structure theorem about $\mathbf{Z}$ says that all its subgroups are of form $n\mathbf{Z}$.

Theorem 2.2. *Given a subgroup H of $\mathbf{Z}$ other than $\{0\}$, we define n as the smallest positive element of H. We have $H = n\mathbf{Z}$.*

Proof. Since $n \in H$, it is clear that $n\mathbf{Z} \subseteq H$, since H must be closed by addition and inversion.

To show that $H \subseteq n\mathbf{Z}$, we take $a \in H$ arbitrary and we show that a is a multiple of n. For this, we make the Euclidean division of a by n and obtain q and r such that $a = qn + r$ and $0 \leq r < n$. Since H is closed and since $a, n \in H$, we must have $r = a - qn \in H$. Since n is the smallest positive element of H and $0 \leq r < n$, we must have $r = 0$. So, $a = qn$: a is a multiple of n. □

Here is a consequence.

Theorem 2.3. *Given a (multiplicative) finite group G and $g \in G$, we define the* order *of g as being the smallest positive integer n such that $g^n = 1$. For any integer i, $g^i = 1$ is equivalent to i being a multiple of n.*

Proof. We define H as the set of all integers i such that $g^i = 1$. Clearly, H is closed by addition (if $g^i = 1$ and $g^j = 1$, then $g^{i+j} = g^i g^j = 1$) and by inversion (if $g^i = 1$, then $g^{-i} = 1/g^i = 1$). So, H is a subgroup of $\mathbf{Z}$. Using the previous theorem, we have $H = n\mathbf{Z}$ where n is the smallest positive element of H, i.e., the order of g. Consequently, $g^i = 1$ is equivalent to $i \in H$ which is equivalent to $i \in n\mathbf{Z}$ which is equivalent to i being a multiple of n. □

Theorem 2.4. *Given a (multiplicative) finite group G and $g \in G$, the subgroup generated by g is $\langle g \rangle = \{g^0, g^1, \ldots, g^{n-1}\}$ where n is its cardinality.*

Proof. We know that $\langle g \rangle = \{\ldots, g^{-2}, g^{-1}, 1, g^1, g^2, \ldots\}$.

Clearly, $\{g^0, g^1, \ldots, g^{n-1}\} \subseteq \langle g \rangle$.

To show the inclusion in the other direction, we let $a \in \langle g \rangle$ be arbitrary and we want to show that $a \in \{g^0, g^1, \ldots, g^{n-1}\}$. The element a can be written $a = g^i$ with $i \in \mathbf{Z}$. Let $i = qn + r$ be the Euclidean division of i by n. We have $g^i = g^{qn+r} = g^r$ with $0 \leq r < n$. So, $a \in \{g^0, g^1, \ldots, g^{n-1}\}$.

Thus, $\langle g \rangle = \{g^0, g^1, \ldots, g^{n-1}\}$.

To show that the cardinality of $\langle g \rangle$ is n, we just observe that for $0 \leq i, j < n$, $g^i = g^j$ implies that $g^{i-j} = 1$, so that $i - j$ is a multiple of n. But since $0 \leq i, j < n$, this implies that $i = j$. □

Actually $g^i = g^j$ is equivalent to $i \equiv j \pmod n$. Since exponents of g can be taken modulo n, we have indeed an isomorphism between $\mathbf{Z}_n$ and $\langle g \rangle$ by mapping i to g^i.

The *order* of a finite group is its cardinality. (So, the order of an element is the order of the subgroup that it spans.) The *Lagrange Theorem* says that in a finite group, the order of any element must be a factor of the order of the group. Consequently, in a finite multiplicative group of order n, for every group element g, we have $g^n = 1$.

A simple consequence is that if G has a prime order p, then all its elements except 1 have order p, so they are generators of G.

In 1976, Diffie and Hellman published the foundations of public key cryptography [32]. This paper included a *key exchange protocol*. The Diffie-Hellman key exchange protocol is used so that two people, Alice and Bob, who did not originally share any secret and who are communicating over a public channel, can set up a symmetric secret key. This works in a group generated by some g. Alice first secretly picks a random x computes $X = g^x$ and sends X to Bob. Bob does the same. He secretly picks a random y computes $Y = g^y$ and sends Y to Alice. Then, Alice computes $K = Y^x$ which should match the result that Bob gets by computing $K = X^y$. This would be $K = g^{xy}$. Interestingly, for some groups, there exists no efficient algorithm to compute K from X and Y although there is an algorithm to compute g^x from g and x. This is why we have this key exchange protocol.

2.3 Algorithms for Big Numbers

The algorithms that children learn at school to add and multiply "big" decimal numbers can generalize in any other basis (e.g., binary, hexadecimal). The addition algorithm has a linear complexity in terms of the length of the operands. The multiplication algorithm has a quadratic complexity.

The multiplication algorithm is in fact not a *group multiplication* algorithm: it is the multiplication of a (big) integer by an element of a (additive) group. We can actually use the same algorithm to raise an element of a (multiplicative) group to some integral power. More generally, it works on any *monoid* instead of a group: we do not need every element to have an inverse.

Given a monoid (using additive notations), if we want to multiply a monoid element a by an integer e, we can decompose e in binary and scan the bits of e while doubling and adding terms. This is called the *double-and-add algorithm*. We have different algorithms based on the scanning direction: there are the right-to-left and the left-to-right algorithms. The algorithm scanning e from right to left is as follows:

Input: $a \in G$ and an integer e of ℓ bits written as $e = \sum_{i=0}^{\ell-1} e_i 2^i$ with $e_i \in \{0, 1\}$
Output: $b = e \cdot a$

$x \leftarrow 0$
$y \leftarrow a$
for $i = 0$ to $\ell - 1$ **do**
  if $e_i = 1$ **then**
    $x \leftarrow x + y$
  end if
  $y \leftarrow y + y$
end for
$b \leftarrow x$

By scanning from left to right, we obtain the following algorithm:

Input: $a \in G$ and an integer e of ℓ bits written as $e = \sum_{i=0}^{\ell-1} e_i 2^i$ with $e_i \in \{0, 1\}$
Output: $b = e \cdot a$

$x \leftarrow 0$
for $i = \ell - 1$ to 0 **do**
  $x \leftarrow x + x$
  if $e_i = 1$ **then**

$x \leftarrow x + a$
end if
end for
$b \leftarrow x$

If instead we use a monoid with multiplicative notations, we can just replace the $+$ by $\times$ in the previous algorithms and obtain the *square-and-multiply* algorithms. Assuming that a monoid multiplication has a complexity $\mathcal{O}(T)$, raising an element to the power e has a complexity $\mathcal{O}(T \log e)$. This is much lower than $T \times e$.

2.4 $\mathbf{Z}_n$: The Ring of Residues Modulo n

Definition 2.5. *A* ring *is a set with two two-input operations:* $+$ *and* $\times$*. It must be such that*

- *the set together with* $+$ *is an Abelian group,*
- *the set is closed under multiplication,*
- $\times$ *is associative,*
- *there is a neutral element* 1 *for* $\times$*,*
- $\times$ *is distributed over* $+$*.*

The ring is commutative *if* $\times$ *is further commutative.*

Not every element in a ring has a multiplicative inverse. Those which have one are called *units*. Given a ring R, R^* denotes the set of units. It forms a group with $\times$ and is often called the *multiplicative group* of R.

As for groups, there are several ring constructors: subrings, ring products, ring powers, quotient. Actually, the notion of subring is not so useful. We favor the notion of *ideal* of a ring, which is a subgroup for $+$ that is stable by multiplication by any ring element (instead of just being stable by any subring element). For instance, $n\mathbf{Z}$ is an ideal of $\mathbf{Z}$ as it is a subgroup and the product of any $n\mathbf{Z}$ element by any $\mathbf{Z}$ element is still in $n\mathbf{Z}$. We can quotient rings by ideals. We have already defined $\mathbf{Z}_n$ in an ad-hoc way. The cerebral way consist of making a quotient of the ring $\mathbf{Z}$ by its ideal $n\mathbf{Z}$. So, $\mathbf{Z}_n$ is actually a ring when considering the addition and multiplication reduced modulo n. The group of units of $\mathbf{Z}_n$ is $\mathbf{Z}_n^*$. It is a multiplicative group. The order of $\mathbf{Z}_n^*$ is denoted by $\varphi(n)$.

For n large, of size ℓ bits, we thus have

- an algorithm of complexity $\mathcal{O}(\ell)$ to compute additions in $\mathbf{Z}_n$;
- an algorithm of complexity $\mathcal{O}(\ell^2)$ to compute multiplications in $\mathbf{Z}_n$;
- an algorithm of complexity $\mathcal{O}(\ell^2 \log e)$ to raise to the power e in $\mathbf{Z}_n$.

We can also compute inverses modulo n, with complexity $\mathcal{O}(\ell^2)$. Actually, the following algorithm computes (as the result u) the inverse of a modulo b, when it has an inverse:

Input: a and b, two integers of at most ℓ bits
Output: d, u, v such that $d = au + bv = \mathsf{gcd}(a, b)$
Complexity: $\mathcal{O}(\ell^2)$

$\vec{x} \leftarrow (a, 1, 0)$, $\vec{y} \leftarrow (b, 0, 1)$
while $y_1 > 0$ **do**
make an Euclidean division $x_1 = qy_1 + r$
do simultaneously $\vec{x} \leftarrow \vec{y}$ and $\vec{y} \leftarrow \vec{x} - q\vec{y}$
end while
$(d, u, v) \leftarrow \vec{x}$

The vectors x and y have three coordinates, where x_1 and y_1 denote their first coordinate, respectively. If we consider the algorithm computing only the first coordinate, we recognize the *Euclid algorithm* which computes $\mathsf{gcd}(a,b)$. It will thus be the result in d at the end of the algorithm. We can easily see that all vectors are of form (α, β, γ) such that $\alpha = a\beta + b\gamma$. Consequently, the result is such that $d = au + bv$. The algorithm is called the *extended Euclid algorithm.*

Theorem 2.6. *A number a is invertible modulo a number b if and only if* $\mathsf{gcd}(a,b) = 1$.

Proof. If a is invertible modulo b, then $ax - qb$ is always divisible by $\mathsf{gcd}(a,b)$. We take x a being the inverse of a and q be the integral quotient of ax by b so that $ax - qb = 1$. We obtain that 1 is divisible by $\mathsf{gcd}(a,b)$. Hence, $\mathsf{gcd}(a,b) = 1$.

Conversely, if $\mathsf{gcd}(a,b)$, the extended Euclid algorithm produces u and v such that $1 = au+bv$. Consequently, $(au) \bmod b = 1$, thus a is invertible. Therefore, $\mathsf{gcd}(a,b) = 1$ is equivalent to a being invertible modulo b. □

2.5 Orders in a Group

Definition 2.7. *Given x in a (multiplicative) group G, the* order *of an element $x \in G$ is the smallest positive integer m such that $x^m = 1$.*

We have seen (see Th. 2.2) that every subgroup of $\mathbf{Z}$ can be written in the form $n\mathbf{Z}$, where n is the smallest non-negative element of the subgroup. So, the set of integers i such that $x^i = 1$ forms a subgroup of $\mathbf{Z}$. If we write it as $m\mathbf{Z}$ as in the above result, where m is the smallest positive element of this subgroup, we call *m the order of x in G.*

Definition 2.8. *Given a (multiplicative) group G, the* exponent *of G is the smallest positive integer λ such that $x^\lambda = 1$ for all $x \in G$.*

Indeed, the set of integers i such that $x^i = 1$ for all $x \in G$ forms also a subgroup of $\mathbf{Z}$. We further know that it contains the order of G, due to the Lagrange Theorem. If we write it as $\lambda\mathbf{Z}$ as in the above result, λ is *the exponent of G*. We note that λ must be part of the previously defined subgroup for any x, so λ must be a multiple of the order m of x. This is true for all x. Furthermore, due to the Lagrange Theorem, we deduce that the order of G must be a multiple of λ, which is itself a multiple of m.

When $G = \mathbf{Z}_n^*$, we write $\lambda = \lambda(n)$ where λ defines *the Carmichael function.* We obtain that for all $x \in \mathbf{Z}_n^*$, $\varphi(n)$ is a multiple of $\lambda(n)$, which is itself a multiple of the order of every x in $\mathbf{Z}_n^*$.

We will see that the general formula to compute φ is given by

$$\varphi(p_1^{a_1} \times \cdots \times p_r^{a_r}) = (p_1 - 1)p_1^{a_1 - 1} \times \cdots \times (p_r - 1)p_r^{a_r - 1}$$

when the p_i's are pairwise different prime numbers and the a_i's are positive integers.

There is a general formula to compute $\lambda(n)$ similar to the one of $\varphi(n)$:

$$\lambda(p_1^{a_1} \times \cdots \times p_r^{a_r}) = \mathsf{lcm}\left(\lambda(p_1^{a_1}), \ldots, \lambda(p_r^{a_r})\right)$$

where the p_i's are pairwise different prime numbers and the a_i's are positive integers. For p prime and α integer, we have

$$\lambda(p^\alpha) = \begin{cases} \varphi(p^\alpha) & \text{if } p > 2 \text{ or } \alpha \leq 2 \\ \frac{1}{2}\varphi(p^\alpha) & \text{if } p = 2 \text{ and } \alpha \geq 3 \end{cases}$$

Checking a generator in groups with known order. In a group G whose order n is known, it is not so easy to check whether an element $g \in G$ is a generator. When the factorization of n into primes $n = p_1^{a_1} \times \cdots \times p_r^{a_r}$ is known, we can efficiently do it by checking, for all i, that $g^{n/p_i} \neq 1$. This is indeed a necessary condition (otherwise, the order would be smaller). Since the order of n must be a factor of n, we can see that, if not equal to n, it must be a factor of some n/p_i. So, the conditions together are sufficient for g to be a generator. We obtain the following algorithm:

Input: an element g in an Abelian cyclic group of order with known factorization $n = p_1^{a_1} \times \cdots \times p_r^{a_r}$

for $i = 1$ to r **do**
  $y \leftarrow g^{n/p_i}$
  if $y = 1$ **then**
    output "g is not a generator" and stop
  end if
end for
output "g is a generator"

If g is provided by some untrusted party, g could have been maliciously selected and there is essentially no better way to check that g is a generator. So, it is required to know the factorization of n.

When g can be trusted to have been randomly selected, it is unlikely that $g^{n/p_i} = 1$ for any large p_i. So, it is enough to check the condition for every small p_i. Since it is easy to find the small prime factors of n, this is doable by knowing only n. To go back to the untrusted g case, a way to convince that g was randomly selected could be to provide the seed of the pseudorandom generator which generated g.

In more details, we consider the following algorithm:

Input: an element g in an Abelian cyclic group of known order n, a parameter B

find the list $p_1, \ldots, p_s$ of all prime factors of n which are less than B
for $i = 1$ to s **do**
  $y \leftarrow g^{n/p_i}$
  if $y = 1$ **then**
    output "g is not a generator" and stop
  end if
end for
output "g may be a generator"

Clearly, if the output is "g is not a generator", the statement is correct. Then, we show that when g is uniformly selected in the group, the probability that g is not a generator given that the output is "g may be a generator" is bounded by $\frac{\log n}{B \log B}$. With a relatively small B, we can find the list $p_1, \ldots, p_s$ easily and we can thus have a good confidence that the output is correct.

We define the event E_i that $g^{\frac{n}{p_i}} = 1$ when g is uniformly distributed in G. To prove correctness, we first think of G with another representation. We already know that G is cyclic, so it is isomorphic to Z_n. If we take $n = p_1^{a_1} \times \cdots \times p_r^{a_r}$ the complete factorization into primes p_i with $p_i \neq p_j$ and $a_i \geq 1$, by using the Chinese Remainder Theorem (to be seen later), we obtain that G is isomorphic to $\mathbf{Z}_{p_1^{a_1}} \times \cdots \times \mathbf{Z}_{p_r^{a_r}}$. So, picking g uniformly in G is equivalent to picking some independent x_j uniformly in $\mathbf{Z}_{p_j^{a_j}}$ for $j = 1, \ldots, r$. So, E_i is equivalent to having $\frac{n}{p_i} x_j \bmod p_j^{a_j} = 0$ for $j = 1, \ldots, r$. Since $p_j^{a_j}$ divides $\frac{n}{p_i}$ for all $j \neq i$, this is always the case. But for $j = i$, this is equivalent $p_i^{a_i - 1} x_i \bmod p_i^{a_i} = 0$. This is equivalent to $x_i \bmod p_i = 0$. So, the probability of E_i is equal to $\frac{1}{p_i}$. We further note that the events E_i are independent.

Let Maybe be the event that the output is "g may be a generator". The event Maybe is the complement of $E_1 \cup \cdots \cup E_s$. The event $\mathsf{NotGenerator}$ that g is not a generator is $E_1 \cup \cdots \cup E_t$. So,

$$\Pr[\mathsf{NotGenerator} | \mathsf{Maybe}] = \Pr[E_{s+1} \cup \cdots \cup E_r | \neg(E_1 \cup \cdots \cup E_s)] \leq \sum_{i=s+1}^{r} \Pr[E_i | \neg(E_1 \cup \cdots \cup E_s)]$$

Since E_i and $E_1 \cup \cdots \cup E_s$ are independent, for $i > s$ we have $\Pr[E_i | \neg(E_1 \cup \cdots \cup E_s)] = \Pr[E_i] = \frac{1}{p_i} \leq \frac{1}{B}$. So,

$$\Pr[\mathsf{NotGenerator} | \mathsf{Maybe}] \leq \frac{r - s}{B} \leq \frac{\log n}{B \log B}$$

where the last inequality comes from $n \geq p_{s+1}^{a_{s+1}} \cdots p_r^{a_r} \geq p_{s+1} \cdots p_r \geq B^{r-s}$.

As another application, we can *find* a generator of G by picking g at random and checking that $g^{n/p_i} \neq 1$ for every small prime factor p_i of n. The algorithm is as follows:

Input: the order n of an Abelian cyclic group and a parameter B

find the list $p_1, \ldots, p_s$ of all prime factors of n which are less than B
repeat
    pick a random g in the group
    $b \leftarrow \mathsf{true}$
    for $i = 1$ to s **do**
        $y \leftarrow g^{n/p_i}$
        if $y = 1$ **then**
            $b \leftarrow \mathsf{false}$
        end if
    end for
until b
output g

The probability that the output is not a generator is bounded by $\frac{\log n}{B \log B}$.

This algorithm can be used to find a generator of $\mathbf{Z}_p^*$ when p is prime with $n = p - 1$.

2.6 The $\mathbf{Z}_p$ Field

Definition 2.9. *A* field *is a ring in which every non-zero element has a multiplicative inverse.*

Whenever p is a prime number, $\mathbf{Z}_p$ is a field. Indeed, every non-zero element must have its gcd with p equal to 1, so it must be invertible. So, $\mathbf{Z}_p^* = \{1, \ldots, p-1\}$. It has order $p - 1$. Due to the Lagrange Theorem, we deduce that for all $x \in \mathbf{Z}_p^*$, we have $x^{p-1} \bmod p = 1$. This is called the *Little Fermat Theorem*.

Theorem 2.10 (Little Fermat Theorem). *If p is a prime number, for all integer x coprime with p, we have $x^{p-1} \bmod p = 1$.*

We could further show that $\mathbf{Z}_p^*$ is cyclic. I.e., it has a generator, as a multiplicative group.

Theorem 2.11. *If p is a prime number, $\mathbf{Z}_p^*$ is a cyclic group.*

Given a group G of order n and an element g generating G, the *discrete logarithm problem* is specified by a group element y. It consists of finding an integer x such that $g^x = y$.

More precisely, we define this computational problem in terms of a *game*. This game depends on some security parameter λ (for instance, the bitlength of the group order) and is specified by a sequence of steps. It starts by settting up some parameters (here: the group we are interested in) then by initializing the game (here: by chosing the instance X the logarithm of which must be found). Then, an *adversary* is given the instance and what he has access to and must return something. Finally, there is a winning condition (here: that the returned value is a discrete logarithm of X).

$\mathsf{DL}(\lambda)$:

$\mathsf{Setup}(1^\lambda) \to (\mathsf{group}, q, g)$
pick $x \in \mathbf{Z}_q$
$X \leftarrow g^x$
$\mathcal{A}(\mathsf{group}, q, g, X) \to x'$
return $1_{X = g^{x'}}$

The *advantage* of the game is often simply defined as the probability to win. (Sometimes, we subtract the probability of a trivial winning probability.) We say that the discrete logarithm problem is hard with respect to Setup if, for every probabilistic polynomial-time algorithm $\mathcal{A}$, the advantage is a negligible function of the security parameter λ.

The notion of "negligible" function is asymptotic. We say that $f(\lambda)$ is negligible when $\lambda \to +\infty$ if for every positive integer n, we have $f(\lambda) = \mathcal{O}(\lambda^{-n})$.

In some groups, this problem is easy. For instance, in the group $\mathbf{Z}_n$ (for which we have to translate the above definition, since $\mathbf{Z}_n$ has some additive notation), it is easy to find x such that $(gx) \bmod n = y$. We just have to divide y by g modulo n.

In any group of order bounded by B, the Baby-Step Giant-Step algorithm [76] solves the discrete logarithm problem in $\mathcal{O}(\sqrt{B})$ group operations.

In the group $\mathbf{Z}_p^*$, the discrete logarithm problem is believed to be hard.

The discrete logarithm problem is easy to solve if we can do computations on a quantum computer (this is the Shor algorithm [78]). It is also easy if the group order n has only small prime factors (this is the Pohlig-Hellman algorithm [63]). For $\mathbf{Z}_p^*$, the best known algorithm is GNFS [56, 65] and runs with a pretty large complexity

$$e^{\left(\sqrt[3]{\frac{64}{9}}+o(1)\right)(\ln n)^{\frac{1}{3}}(\ln\ln n)^{\frac{2}{3}}}$$

We should however be careful with what this figure suggests for security. Indeed, this algorithm spends most of its time to precompute the tables which only depend on the selection of the group and the generator. As many such parameters are very popular (as coming from standard), it is realistic to believe that some agencies may have spent some time to run this precomputation. After the precomputation is done, the attack itself works with a lower complexity of

$$e^{\left(\sqrt[3]{3+o(1)}\right)(\ln n)^{\frac{1}{3}}(\ln\ln n)^{\frac{2}{3}}}$$

The difference can be quite large as reported by Adrian et al. [11] in this table:

p **length** (bits)	**precomputation** (core-time)	**attack** (core-time)
512	10.2 years	10 minutes
768	36 500 years	2 days
1 024	45 000 000 years	30 days

(This was computed by extrapolating some experimental values.) So, after spending a year of precomputation with 28 500 cores, computing a discrete logarithm in $\mathbf{Z}_p^*$ with a given prime p of 768 bits will only take 2 days. This was used to mount the Logjam attack on TLS [11] in 2015.

For instance, SSH2 uses $p = 2^{1024} - 2^{960} - 1 + 2^{64}\lfloor 2^{894}\pi + 129093\rfloor$, $g = 2$, and $q = \frac{p-1}{2}$. (Both p and q are prime, and g has order q in $\mathbf{Z}_p$.) So, if one can invest 45 million core.years of computation, any discrete logarithm for SSH2 could be computed in one core.month.

2.7 The Diffie-Hellman Key Exchange, Concretely

In the specified Diffie-Hellman key exchange protocol [32], there are a few problems to be avoided:

- the existence of subgroups, in general, creates security problems and should be avoided, by taking groups of prime order;
- the existence of the trivial subgroup $\{1\}$ cannot be avoided and a specific countermeasure must be taken in the protocol;
- the representation of the value $K = g^{xy}$ may have some bad distribution and it cannot be used directly as a symmetric key.

For this, we use a group whose order is a prime number q. The numbers x and y shall be selected in $\mathbf{Z}_q^*$ with a uniform distribution. This way, X and Y are uniformly distributed in the group without 1. Alice resp. Bob must verify that Y resp. X is different from 1, i.e. that they are not in the subgroup $\{1\}$. Finally, g^{xy} goes through a specific algorithm called a *Key Derivation Function (KDF)* to generate a useable key K. Here is the complete protocol:

$$
\begin{array}{rcl}
\textbf{Alice} & & \textbf{Bob} \\
& & \\
\text{pick } x \in \mathbf{Z}_q^*,\ X \leftarrow g^x & \xrightarrow{\quad X \quad} & \text{if } X \notin \langle g \rangle - \{1\}, \text{ abort} \\
\text{if } Y \notin \langle g \rangle - \{1\}, \text{ abort} & \xleftarrow{\quad Y \quad} & \text{pick } y \in \mathbf{Z}_q^*,\ Y \leftarrow g^y \\
K \leftarrow \mathsf{KDF}(Y^x) & & K \leftarrow \mathsf{KDF}(X^y) \\
\text{output: } K & (K = \mathsf{KDF}(g^{xy})) & \text{output: } K
\end{array}
$$

The protocol may resist to *passive attacks*, where the adversary only sees the messages X and Y but does not modify them. The security holds if the *Computational Diffie-Hellman problem* (CDH) is hard. Given a group, a generator g, and its order, this problem is specified by a pair (X, Y) of group elements. The goal is to compute g^{xy} where x and y are such that $X = g^x$ and $Y = g^y$. Like for the discrete logarithm problem, this one may be easy or hard to solve depending on the group. Typically, we would select a subgroup of $\mathbf{Z}_p^*$ of prime order q, or an elliptic curve (to be seen in a future chapter). The corresponding game is as follows:

$\mathsf{CDH}(\lambda)$:

$\mathsf{Setup}(1^\lambda) \to (\mathsf{group}, q, g)$
pick $x, y \in \mathbf{Z}_q$
$X \leftarrow g^x$, $Y \leftarrow g^y$
$\mathcal{A}(\mathsf{group}, q, g, X, Y) \to K$
return $1_{K=g^{xy}}$

Another important problem is the *Decisional Diffie-Hellman problem* (DDH) in which the adversary must recognize if a proposed solution to the computational problem is correct or not. More precisely, the game is as follows:

$\mathsf{DDH}(\lambda, b)$:

$\mathsf{Setup}(1^\lambda) \to (\mathsf{group}, q, g)$
pick $x, y, z \in \mathbf{Z}_q$
if $b = 1$ **then** $z \leftarrow xy$
$X \leftarrow g^x$, $Y \leftarrow g^y$, $Z \leftarrow g^z$
$\mathcal{A}(\mathsf{group}, q, g, X, Y, Z) \to t$
return t

The advantage is no longer the probability to return 1. It is the difference between $b = 1$ and $b = 0$ which is an additional parameter in the game:

$$\mathsf{Adv}_{\mathcal{A}}(\lambda) = \Pr[\mathsf{DDH}(\lambda, 1) \to 1] - \Pr[\mathsf{DDH}(\lambda, 0) \to 1]$$

Note that the hardness of the computational Diffie-Hellman problem implies the hardness of the discrete logarithm problem in the same group. Indeed, if we could solve the discrete logarithm problem, we would compute x from $X = g^x$, then Y^x.

One example of concrete implementation of the Diffie-Hellman problem is given by RFC 2631 [68]. There, the KDF function is a specific algorithm based on the SHA1 hash function (to be seen in a future chapter). The result is used to encrypt a key which is used in further exchanges. This is called *key wrapping*.

In RFC 2631, setting up the group means to define two prime numbers p and q and one integer g such that $\langle g \rangle$ is a subgroup of order q in $\mathbf{Z}_p^*$. One method to check that (p, q, g) is a valid triplet of parameters consists of checking that p and q are prime, that $g \bmod p \neq 1$, and that $g^q \bmod p = 1$. Indeed, in such case, it is guaranteed that g has order q in $\mathbf{Z}_p^*$. Later, to check that X and Y are in the subgroup generated by g, we just have to check that $X^q \bmod p = 1$ and $Y^q \bmod p = 1$, respectively. This is due to the following result which essentially says that there is a unique subgroup of $\mathbf{Z}_p^*$ of order q. So, numbers g, X, and Y of order q are necessarily in the same subgroup $\langle g \rangle$.

Theorem 2.12. *If p and q are prime numbers and if $g \bmod p \neq 1$, $g^q \bmod p = 1$, then $\langle g \rangle$ is the set of all $x \in \mathbf{Z}_p$ such that $x^q \bmod p = 1$. It is a subgroup of $\mathbf{Z}_p^*$ of order q.*

Proof. Due to the assumptions, g is clearly an element of $\mathbf{Z}_p^*$ of order q and q divides $p-1$.

The point that $\langle g\rangle \subseteq \{x \in \mathbf{Z}_p; x^q \bmod p = 1\}$ is trivial: since $g^q \bmod p = 1$, $x = g^u$ is such that $x^q \bmod p = 1$.

To show the converse, we take $x \in \mathbf{Z}_p$ such that $x^q \bmod p = 1$. We want to show that x is in $\langle g\rangle$. Clearly, x^{q-1} is the inverse of x in $\mathbf{Z}_p^*$, so $x \in \mathbf{Z}_p^*$. We use the fact that $\mathbf{Z}_p^*$ is cyclic and take a generator θ of this group. Since it is a generator, we can write $g = \theta^a$ and $x = \theta^b$. Due to $g^q \bmod p = 1$, we have $aq \bmod (p-1) = 0$. Similarly, $bq \bmod (p-1) = 0$. We can thus write $a = a'\frac{p-1}{q}$ and $b = b'\frac{p-1}{q}$ for some integers a' and b'. Since $g \bmod p \neq 1$, we have $1 \leq a' < q$. So, a' is invertible modulo q (remember that q is prime). Let c be the inverse of a' modulo q. We have

$$g^{b'c} \equiv \theta^{ab'c} \equiv \theta^{a'bc} \equiv x^{a'c} \pmod p$$

Since we can write $a'c = 1 + kq$ for some integer k, we deduce that $g^{b'c} \equiv x \pmod p$. So, $x \in \langle g\rangle$. □

The plain Diffie-Hellman protocol does not authenticate the ephemeral public keys. Hence, a man-in-the-middle can replace them by others and run separate protocols with both participants. This is an *active attack* which is not avoided by this protocol. However, if the participant succeed to communicate directly without the man-in-the-middle interfering, they may realize that they did not obtain the same key. A more devastating active attack is one which make the participants receive the same key, which the adversary can compute as well. This is possible when there are small subgroups. Hence, we should avoid using groups which have small subgroup. The standard method is to use groups of prime order.

2.8 The ElGamal Public-Key Cryptosystem

In a *Public-Key Cryptosystem (PKC)*, we have three algorithms:

- a *key generation algorithm*, which is a pseudorandom generator producing a key pair $(\mathsf{pk}, \mathsf{sk})$, pk being publicly revealed while sk being private;
- an *encryption algorithm*, which could be a randomized algorithm producing a ciphertext given a plaintext and a public key pk;
- a *decryption algorithm*, being deterministic, and reconstructing the plaintext from the ciphertext and the private key sk.

When the encryption is probabilistic, the ciphertext is often larger than the plaintext, and encrypting the same plaintext several times produces different ciphertexts. We only require that the decryption reconstructs the correct plaintext.

One way to transform the Diffie-Hellman key exchange protocol [32] into a PKC consists of treating Bob's key y as a long-term private key and $Y = g^y$ as his public key. Then, Alice would generate x, compute $X = g^x$, and encrypt her message by using a symmetric encryption scheme based on the key $K = \mathsf{KDF}(Y^x)$. The ciphertext would consists of X and the encrypted message. Bob would recover the message by doing the symmetric decryption with the key $K = \mathsf{KDF}(X^y)$. This type of construction is typical for what we call *hybrid encryption*. We combine public-key cryptography, which is used to derive a symmetric key K (the Diffie-Hellman protocol, in our case), with symmetric encryption. The ElGamal cryptosystem follows this idea but uses no KDF and takes one-time-pad (over the group) as a symmetric encryption.

In the *ElGamal public-key cryptosystem* [37], there is a common group $\langle g\rangle$ of order n. To generate a key pair, we pick $\mathsf{sk} = x \in \mathbf{Z}_n$ and compute $\mathsf{pk} = y = g^x$. A message must be a group element $m \in \langle g\rangle$. To encrypt m, we pick $r \in \mathbf{Z}_n$, compute $u = g^r$, $v = my^r$, and the ciphertext is (u, v). (See Fig. 2.1.) To decrypt (u, v), we compute $m = vu^{-x}$. Clearly, this construction is correct as

$$vu^{-x} = my^r g^{-rx} = mg^{rx}g^{-rx} = m$$

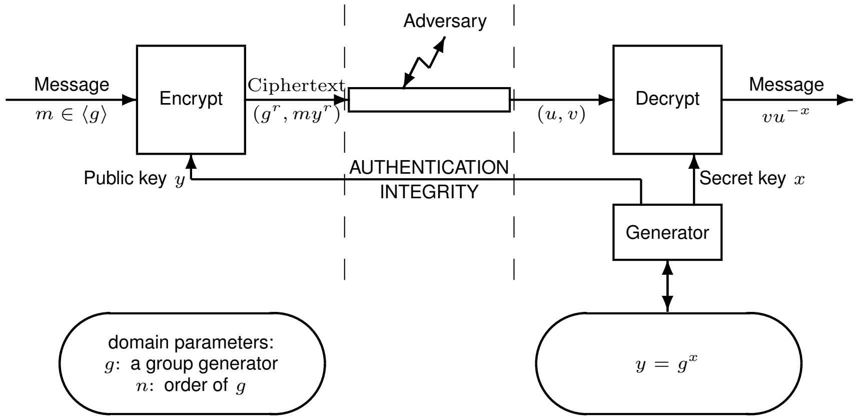

Figure 2.1: The ElGamal Cryptosystem

Once the domain parameters are selected (i.e., the group itself), and since it can be common to all users, the key generation is pretty simple: an exponential computation in the group. Encryption and decryption have a similar cost.

Fig. 2.1 shall be understood as follows: the domain parameters g and n are common and assumed to be reliably known to all participants. There is an initial secure transmission of the public key y from the receiver to the sender. The security of this transmission must preserve the authenticity and integrity of y but not necessarily the confidentiality. The ciphertext is transmitted through an insecure communication channel.

The security of the ElGamal cryptosystem comprises two different problems: the decryption problem and the key recovery problem. They are defined as follows:

EGKR (ElGamal Key Recovery Problem)

$\mathsf{Setup}(1^\lambda) \to (\mathsf{group}, n, g)$
$\mathsf{Gen}(\mathsf{group}, n, g) \to (y, x)$ $\quad\triangleright$ pick $x \in \mathbf{Z}_n$, $y = g^x$
$\mathcal{A}(\mathsf{group}, n, g, y) \to x'$
return $1_{x=x'}$

EGD (ElGamal Decryption Problem)

$\mathsf{Setup}(1^\lambda) \to (\mathsf{group}, n, g)$
$\mathsf{Gen}(\mathsf{group}, n, g) \to (y, x)$ $\quad\triangleright$ pick $x \in \mathbf{Z}_n$, $y = g^x$
pick $\mathsf{pt} \in \langle g \rangle$ $\quad\triangleright$ pick $\mathsf{pt} \in \langle g \rangle$
$\mathsf{Enc}(y, \mathsf{pt}) \to (u, v)$ $\quad\triangleright$ pick $r \in \mathbf{Z}_n$, $u = g^r$, $v = \mathsf{pt} \cdot y^r$
$\mathcal{A}(\mathsf{group}, n, g, y, u, v) \to m$
return $1_{m=\mathsf{pt}}$

Clearly, key recovery is equivalent to the discrete logarithm problem. We show below that the decryption problem is equivalent to the computational Diffie-Hellman problem. So, we need the computational Diffie-Hellman problem to be hard to have security.

There are other tricky things to say about the security of the ElGamal cryptosystem, but this will be left for another course.

To show that the ElGamal decryption problem is equivalent to the computational Diffie-Hellman problem, we use the notion of *Turing reduction* which can compare two problems. If we have an oracle (which is used like a subroutine in a program) to solve the computational Diffie-Hellman problem, then we can make an algorithm which will solve the ElGamal decryption problem and vice versa. Let assume that G is fixed and that the oracle takes X and Y as input and gives K as output, with $X = g^x$, $Y = g^y$, $K = g^{xy}$. Then, given a public key y and a ciphertext (u, v), we can set $X = u$ and $Y = y$, query the oracle, and compute v/K. We can easy see that when (u, v) is the encryption of m, i.e. that $u = g^r$ and $v = my^r$, then v/K is actually m. So, this solved the ElGamal decryption problem given the Diffie-Hellman oracle. For the contrary, we assume a decryption oracle such that given y and (u, v) the oracle returns the decryption of (u, v)

with the secret key associated to y. Given X and Y, we can set $u = X$, $y = Y$, and set a random v in the group. After querying the oracle we obtain m and we can easily see that $K = v/m$ is the answer to the computational Diffie-Hellman problem. This way, we show the equivalence between the two problems.

Chapter 3

RSA Cryptography

In this chapter, we recall basic arithmetic notions, such as the Chinese Remainder Theorem. We study primality testing and present the RSA public-key cryptosystem.

3.1 Euler and Other Chinese

We have already seen that $\mathbf{Z}_n^*$ is the group of units of the ring $\mathbf{Z}_n$: this is the group (for the multiplication) of all elements of $\mathbf{Z}_n$ which have a multiplicative inverse. We have also seen that for all $x \in \mathbf{Z}_n$, x is in $\mathbf{Z}_n^*$ if and only if $\mathsf{gcd}(x, n) = 1$. Finally, we have also seen that $\mathbf{Z}_n^*$ contains all elements but 0 (i.e., $\mathbf{Z}_n$ is a field) if and only if n is a prime number.

The *Euler totient function* is the function φ over the positive integers defined by $\varphi(n) = \#\mathbf{Z}_n^*$. I.e., $\varphi(n)$ is the order of the group $\mathbf{Z}_n^*$. Due to the Lagrange theorem, we have the following result.

Theorem 3.1 (Euler Theorem). *Given a positive integer n, for all $x \in \mathbf{Z}_n^*$ we have $x^{\varphi(n)} \bmod n = 1$.*

Another nice application is the extraction of eth roots in $\mathbf{Z}_n^*$, whenever e has an inverse modulo $\varphi(n)$. Indeed, if e has such inverse (i.e., if $\mathsf{gcd}(e, \varphi(n)) = 1$), let $d = e^{-1} \bmod \varphi(n)$. We can easily see that for all $x \in \mathbf{Z}_n^*$, $x^d \bmod n$ is the only eth root of x in $\mathbf{Z}_n^*$. To see this, we can first check that it is a root. Then, we show that any root must be equal to this one. To see that x^d is an eth root of x, we can just raise it to the power e:

$$(x^d)^e \equiv x^{ed} \equiv x^{1+k\varphi(n)} \equiv x \pmod n$$

since $ed = 1 + k\varphi(n)$ for some integer k and since $x^{\varphi(n)}$ is 1. To see that any eth root y must be equal to x, we start from $y^e \equiv x$. Then, since y^e is invertible, y^{e-1}/y^e must be an inverse of y, so $y \in \mathbf{Z}_n^*$. Consequently, $y^{\varphi(n)} \equiv 1$. Now we raise $y^e \equiv x$ to the power d and we obtain that

$$x^d \equiv y^{ed} \equiv y^{1+k\varphi(n)} \equiv y \pmod n$$

So, $y = x^d \bmod n$.

RSA. This is used in the RSA cryptosystem [71]: the public key consists of a modulus n and some exponent e such that $\mathsf{gcd}(e, \varphi(n)) = 1$. The secret key is the inverse d of e modulo $\varphi(n)$. To encrypt an element x of $\mathbf{Z}_n$, we compute $y = x^e \bmod n$. To decrypt, we compute $x = y^d \bmod n$. Note that, so far, we have shown that it works for $x \in \mathbf{Z}_n^*$ but not for any $x \in \mathbf{Z}_n$. We will see this soon. We will also see that by taking $n = pq$, where p and q are two different prime numbers, then we can compute $\varphi(n) = (p-1)(q-1)$. These are consequences of the Chinese Remainder Theorem.

Chinese Remainder Theorem. If m and n are coprime positive integers (i.e., $\mathsf{gcd}(m,n)=1$), we can see the rational computations over $\mathbf{Z}_{mn}$ as equivalent to the same computations over $\mathbf{Z}_m$ and $\mathbf{Z}_n$ in parallel. This is expressed by saying that $x \mapsto (x \bmod m, x \bmod n)$ is a ring isomorphism between $\mathbf{Z}_{mn}$ and $\mathbf{Z}_m \times \mathbf{Z}_n$.

Theorem 3.2 (Chinese Remainder Theorem). *If m and n are coprime positive integers, the function f defined on $\mathbf{Z}_{mn}$ by mapping x to $f(x) = (x \bmod m, x \bmod n)$ is a ring isomorphism between $\mathbf{Z}_{mn}$ and $\mathbf{Z}_m \times \mathbf{Z}_n$.*

For any $a \in \mathbf{Z}_m$ and $b \in \mathbf{Z}_n$, we have

$$f^{-1}(a,b) = \big(an(n^{-1} \bmod m) + bm(m^{-1} \bmod n)\big) \bmod (mn)$$

Proof. Checking that f is a ring homomorphism is straightforward: for the addition, we have

$$f(x+y) = ((x+y) \bmod m, (x+y) \bmod n) = (x \bmod m, x \bmod n) + (y \bmod m, y \bmod n) = f(x)+f(y)$$

where the last additions are in $\mathbf{Z}_m \times \mathbf{Z}_n$. The same goes for the multiplication.

We then show that f is injective by showing that $f(x) = (0,0)$ implies that $x = 0$ in $\mathbf{Z}_{mn}$. Indeed, $f(x) = (0,0)$ implies that both m and n divide x. Since m and n have no prime factor in common, the unique factoring of x into prime factors must split as the product of m, n, and some left over prime numbers. More precisely, we can write $x = mx'$. Let $n = p_1^{\alpha_1} \cdots p_r^{\alpha_r}$ be the unique factorization of n into pairwise different primes p_i. Since n divides x, each $p_i^{\alpha_i}$ divides $x = mx'$. Since n is coprime with m, p_i does not divide m. Hence, $p_i^{\alpha_i}$ divides x'. So, n divides x' and x' can be written $x' = nx''$. Hence, $x = mnx''$ and so mn divides x. We deduce that $x \bmod (mn) = 0$.

Finally, f must be an isomorphism because the two rings have the same order and f is injective.

The explicit formula for f^{-1} is shown by just checking that f applied to the right-hand side matches (a,b): the first component of $f(\big(an(n^{-1} \bmod m) + bm(m^{-1} \bmod n)\big) \bmod (mn))$ is

$$\big(an(n^{-1} \bmod m) + bm(m^{-1} \bmod n)\big) \bmod (mn) \bmod m$$

The modulo mn operation is absorbed by the final modulo m reduction. Then $n(n^{-1} \bmod m)$ reduces to 1 modulo m while $m(m^{-1} \bmod n)$ reduces to 0. So, the first component is a. Similarly, the second component is b. So, we have identified $f^{-1}(a,b)$. $\square$

The Chinese Remainder Theorem is important enough to see another formulation and proof.

Theorem 3.3 (Chinese Remainder Theorem). *Let m and n be coprime positive integers, and let $u = n(n^{-1} \bmod m)$ and $v = m(m^{-1} \bmod n)$. We define $g(a,b) = (au+bv) \bmod mn$. We note that $g(a+im, b+jn) = g(a,b)$ for any integer i and j, so we can define g as a mapping from $\mathbf{Z}_m \times \mathbf{Z}_n$ to $\mathbf{Z}_{mn}$. We have that g is a ring isomorphism.*

Proof. The point that $g(a+im, b+jn) = g(a,b)$ for any integer i and j is straightforward. The point that g is a group homomorphism is trivial, as g is defined with a linear expression. Now, we can see that $g(a,b) \bmod m = a \bmod m$ and $g(a,b) \bmod n = b \bmod n$. So, we can see that g is a bijection. Therefore, it is a group isomorphism. To show that g is a ring isomorphism, it remains to be shown that it is also homomorphic for the multiplication. For that, we can just see that g^{-1} is homomorphic for the multiplication (in the way we did it for f before). $\square$

As a direct consequence, we can see that for any a and b, there exists a unique x (modulo mn) such that $x \equiv a \pmod m$ and $x \equiv b \pmod n$ at the same time. We can use this property, for instance, to deduce from the equalities $x \equiv y \pmod m$ and $x \equiv y \pmod n$ that we in fact have $x \equiv y \pmod{(mn)}$.

Here is another direct consequence.

Theorem 3.4. *Let m and n be coprime positive integers. We have $\varphi(mn) = \varphi(m) \times \varphi(n)$.*

We stress that this holds for $\mathsf{gcd}(m,n) = 1$, i.e. not in general.

Proof. We first observe that the group of units of $\mathbf{Z}_m \times \mathbf{Z}_n$ is the group $\mathbf{Z}_m^* \times \mathbf{Z}_n^*$. Indeed, if (a,b) is invertible in $\mathbf{Z}_m \times \mathbf{Z}_n$, there must be some (c,d) such that $(a,b) \times (c,d) = (1,1)$. As this means that $(ac) \bmod m = 1$ and $(bd) \bmod n = 1$, we obtain that $(a,b) \in \mathbf{Z}_m^* \times \mathbf{Z}_n^*$. Conversely, if $(a,b) \in \mathbf{Z}_m^* \times \mathbf{Z}_n^*$, then $(a^{-1} \bmod m, b^{-1} \bmod n)$ is the inverse of (a,b), so $(a,b) \in (\mathbf{Z}_m \times \mathbf{Z}_n)^*$.

Then, the ring isomorphism between $\mathbf{Z}_m \times \mathbf{Z}_n$ and $\mathbf{Z}_{mn}$ induces a group isomorphism between $(\mathbf{Z}_m \times \mathbf{Z}_n)^*$ and $\mathbf{Z}_{mn}^*$. So, $\mathbf{Z}_m^* \times \mathbf{Z}_n^*$ and $\mathbf{Z}_{mn}^*$ are isomorphic groups. Therefore, they have the same cardinality. I.e., $\varphi(m) \times \varphi(n) = \varphi(mn)$. □

Since we already know that $\varphi(p) = p - 1$ and $\varphi(q) = q - 1$ for two prime numbers p and q, we deduce that when $p \neq q$, we have $\varphi(pq) = (p-1) \times (q-1)$, which was the announced result in the RSA construction [71].

The $\mathbf{Z}_{mn} \approx \mathbf{Z}_m \times \mathbf{Z}_n$ result for m and n coprime generalizes to $\mathbf{Z}_{p_1^{\alpha_1} \times \cdots \times p_r^{\alpha_r}} \approx \mathbf{Z}_{p_1^{\alpha_1}} \times \cdots \times \mathbf{Z}_{p_r^{\alpha_r}}$ for pairwise distinct primes $p_1, \ldots, p_r$. We used this result in Section 2.5 to generate generators of a cyclic group of known order. To compute $\varphi(p_1^{a_1} \times \cdots \times p_r^{a_r})$ we have to count the number of invertible elements in the ring $\mathbf{Z}_{p_1^{\alpha_1} \times \cdots \times p_r^{\alpha_r}}$, so to count the number of invertible elements in each $\mathbf{Z}_{p_i^{\alpha_i}}$ and to multiply them together. In $\mathbf{Z}_{p_i^{\alpha_i}}$, a number is invertible if and only if it is coprime with $p_i^{\alpha_i}$, hence, if and only if it is coprime with p_i. So, we have $p_i^{\alpha_i - 1}$ non-invertible elements. So, $\varphi(p_i^{\alpha_i}) = (p_i - 1)p_i^{\alpha_i - 1}$. We deduce the general formula to compute φ:

$$\varphi(p_1^{a_1} \times \cdots \times p_r^{a_r}) = (p_1 - 1)p_1^{a_1 - 1} \times \cdots \times (p_r - 1)p_r^{a_r - 1}$$

when the p_i's are pairwise different prime numbers and the a_i's are positive integers.

An application of the Chinese Remainder Theorem is the proof that RSA works over $\mathbf{Z}_n$ and not only on $\mathbf{Z}_n^*$.

Theorem 3.5. *Let p and q be two different primes. Let $n = pq$, e be coprime with $\varphi(n)$ and $d = e^{-1} \bmod \varphi(n)$. For all $x \in \mathbf{Z}_n$, we have $x^{ed} \bmod n = x$.*

Therefore, the decryption of the encryption of x is x for all $x \in \mathbf{Z}_n$.

Proof. We know that $(ed) \bmod (p-1) = 1$. So, due to the Little Fermat Theorem (Th. 2.10), we have that $x^{ed} \bmod p = x$ for all $x \in \mathbf{Z}_p^*$. For $x = 0$, we also have $x^{ed} \bmod p = x$. So, we have $x^{ed} \bmod p = x$ for all $x \in \mathbf{Z}_p$. Similarly, we have $x^{ed} \bmod q = x$ for all $x \in \mathbf{Z}_q$. So, x^{ed} matches x modulo p and modulo q. Since p and q are coprime, due to the Chinese Remainder Theorem, x^{ed} matches x modulo n as well: $x^{ed} \bmod n = x$. □

Another interesting application is the RSA acceleration algorithm: to compute $y^d \bmod n$, instead of running the square-and-multiply algorithm with the values y, d, and n, which takes a cubic time in the length of n, we can compute $d_p = d \bmod (p-1)$, $d_q = d \bmod (q-1)$, $y_p = y^{d_p} \bmod p$, $y_q = y^{d_q} \bmod q$, and $f^{-1}(y_p, y_q)$. Clearly, y_p is equal to y^d modulo p, and y_q is equal to y^d modulo q. So, $f^{-1}(y_p, y_q)$ is equal to y^d modulo p and modulo q. So, it is y^d modulo n. Those operations work in quadratic time, except the ones for computing y_p and y_q, which work in cubic time, but with half-size inputs. So, the expected speed-up factor must be close to 4 if we implement the computation this way.

3.2 Primality Testing

Given an integer n, the regular trial division algorithm factors n within $\sqrt{n}$ arithmetic operations, which is huge. It can also be used to check primality, but this is highly inefficient.

Fermat primality test. If n is prime and $b \in \{1, \ldots, n-1\}$, we know that $b^{n-1} \bmod n = 1$. This is the little Fermat Theorem (Th. 2.10). We can use this property to check primality: given n, we pick a random $b \in \{1, \ldots, n-1\}$ then check if $b^{n-1} \bmod n = 1$. This is called the *Fermat test*. The algorithm is as follows:

Input: n, an integer of ℓ bits, and a parameter k

repeat
  pick a random b such that $0 < b < n$
  $x \leftarrow b^{n-1} \bmod n$
  if $x \neq 1$ **then**
    output "n is composite" and stop
  end if
until k iterations are made
output "n may be prime" and stop

Let Prime be the event that n is prime and Maybe be the event that the algorithm return "n may be prime". The Little Fermat Theorem says that $\Pr[\mathsf{Maybe}|\mathsf{Prime}] = 1$: prime numbers will always pass this test. But the remaining question is to bound $\Pr[\mathsf{Maybe}|\neg\mathsf{Prime}]$. Unfortunately, there exist a category of non-prime numbers which may pass it with high probability (but not always) as well: the *Carmichael numbers*. We give below the definition and characterization of Carmichael numbers without a proof.

Theorem 3.6 (Carmichael numbers). *For an integer n, the two following properties are equivalent.*

- *The number n is a product of (at least 2) pairwise different prime numbers p_i such that $p_i - 1$ is a factor of $n - 1$.*
- *The number n is composite and for any b such that $\mathsf{gcd}(b, n) = 1$, we have $b^{n-1} \equiv 1 \pmod{n}$.*

If this happens, we say that n is a Carmichael numbers.

For instance, $n = 561$ is a Carmichael number since $n = 3 \times 11 \times 17$, and for $p \in \{3, 11, 17\}$, we can check that $p - 1$ is a factor of $n - 1$.

When we have a Carmichael number n, we can easily see that whenever $b \in \mathbf{Z}_n^*$, we have $b^{n-1} \bmod n = 1$. Indeed, we have $b^{n-1} \bmod p = 1$ for each prime factor p of n due to the Little Fermat Theorem, since $n - 1$ is a multiple of $p - 1$. Hence, by applying the Chinese Remainder Theorem, we deduce that $b^{n-1} \bmod n = 1$. When $b \notin \mathbf{Z}_n^*$, we cannot obtain $b^{n-1} \bmod n = 1$. So, in that case, we have that $\Pr[\mathsf{Maybe}|n] = \left(\frac{\varphi(n)}{n-1}\right)^k$.

With $n = 949\,631\,589\,089$, we have another Carmichael number. Actually, we have

$$\begin{aligned} n &= 6917 \times 10193 \times 13469 \\ n - 1 &= 2^5 \times 7^3 \times 13 \times 19 \times 37 \times 9467 \end{aligned}$$

Hence

$$\Pr[\mathsf{Maybe}|n] = \left(\frac{\varphi(n)}{n-1}\right)^k = \left(\frac{9464}{9467}\right)^k \approx (1 - 0.000317)^k$$

So, even with many iterations k, the probability that n passes this test is not acceptable. Therefore, we have to extend this primality test to exclude Carmichael numbers.

Miller-Rabin primality test. Given n, we easily rule out the particular cases of $n = 2$ and n even. For n odd, we write $n - 1 = 2^s t$ with t odd. Given a random $b \in \{1, \ldots, n-1\}$, we compute the chain $(b^t \bmod n, b^{2t} \bmod n, \ldots, b^{2^s t} \bmod n)$. (Note that only the last term of the chain is computed in the Fermat test.) If the chain is of form $(1, 1, \ldots, 1)$, or of form $(*, \ldots, *, -1, 1, \ldots, 1)$, we say that n passes the test.

The algorithm is as follows:

Input: n, an integer of ℓ bits, and a parameter k
if $n = 2$ **then**
  output "n is prime" and stop
end if
if n is even **then**

output “n is composite” and stop
end if
write $n = 2^s t + 1$ with t odd
repeat
pick $b \in \{1, \ldots, n-1\}$
$x \leftarrow b^t \bmod n$, $i \leftarrow 0$
if $x \neq 1$ **then**
while $x \neq n-1$ **do**
$x \leftarrow x^2 \bmod n$, $i \leftarrow i+1$
if $i = s-1$ or $x = 1$ **then**
output “n is composite” and stop
end if
end while
end if
until k iterations are made
output “n may be prime” and stop

With the same notations as in the Fermat test, we clearly have $\Pr[\mathsf{Maybe}|\mathsf{Prime}] = 1$: prime numbers always pass this test, as the previous element in the chain must be a square root and that the only square roots of 1 are 1 and -1 in the field $\mathbf{Z}_n$ (this is when n is prime). We will see this in Th. 3.10. We actually have the following result.

Theorem 3.7. *For an odd number $n = 2^s t + 1$ where t is odd, n is a prime number if and only if for all $b \in \mathbf{Z}_n^*$, the chain of $(b^t \bmod n, b^{2t} \bmod n, \ldots, b^{2^s t} \bmod n)$ is either of form $(1, 1, \ldots, 1)$ or of form $(*, \ldots, *, -1, 1, \ldots, 1)$.*

Proof (sketch). When n is prime, the proof is straightforward. To prove the backward direction, we assume that all chains are of form $(1, 1, \ldots, 1)$ or $(*, \ldots, *, -1, 1, \ldots, 1)$. This implies that for all $b \in \mathbf{Z}_n^*$, we have $b^{n-1} \bmod n = 1$. We can show that this implies that either n is prime or that n is a Carmichael number. Then, we can prove that if n is a Carmichael number, there are some $b \in \mathbf{Z}_n^*$ such that the chain is not of the correct form. $\square$

We can even show a more precise result.

Theorem 3.8. *Let n be an odd number $n = 2^s t + 1$ where t is odd. We say that b passes the test if the chain of $(b^t \bmod n, b^{2t} \bmod n, \ldots, b^{2^s t} \bmod n)$ is either of form $(1, 1, \ldots, 1)$ or of form $(*, \ldots, *, -1, 1, \ldots, 1)$. The number n is prime if and only if the cardinality of the set of all $b \in \mathbf{Z}_n^*$ which pass the test is at least $\frac{1}{4}\varphi(n)$.*

So, we deduce that $\Pr[\mathsf{Maybe}|\neg\mathsf{Prime}] \leq 4^{-k}$. So, by having k relatively small, we can rule out composite numbers reliably.

Note that one iteration of this test has a cubic complexity. A composite number is most likely to be ruled out in the very first iterations, so they are eliminated in cubic time. Contrarily, a prime number has to go through all the k iterations before being declared as maybe prime.

Prime number generation. The best way to find a prime number is to pick a random number and try again until it is prime! We have the following important result from number theory.

Theorem 3.9 (Prime Number Theorem). *The probability that a random element selected in the set $\{1, 2, 3, 4, \ldots, N\}$ is prime is asymptotically equivalent to $\frac{1}{\ln N}$.*

So, to pick a prime number of size ℓ, we need $\mathcal{O}(\ell)$ trials. Since we repeat $\mathcal{O}(\ell)$ times the primality test, we want to make sure that for a single test we have $\Pr[\mathsf{Maybe}|\neg\mathsf{Prime}] \leq \frac{\varepsilon}{\ell}$ for some small constant ε. So, we need $k = \frac{1}{2}\log_2 \frac{\ell}{\varepsilon} = \mathcal{O}(\ell)$ due to $\Pr[\mathsf{output:\ maybe\ prime}|\neg\mathsf{prime}] \leq 4^{-k}$. Each of the $\mathcal{O}(\ell)$ trials eliminates the composite number in time $\mathcal{O}(\ell^3)$. The final test takes a bit more time but is not larger than $\mathcal{O}(\ell^4)$. So, we can pick a random prime number of size ℓ within a complexity of $\mathcal{O}(\ell^4)$.

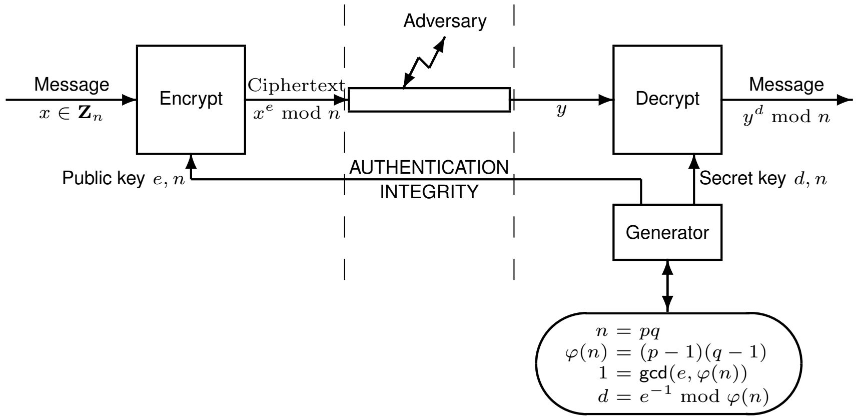

Figure 3.1: The RSA Cryptosystem

3.3 RSA Basics

We describe again the RSA cryptosystem [71]: the key generation consists of generating two different large prime numbers p and q. The public key consists of a modulus $n = pq$ and some exponent e such that $\mathsf{gcd}(e, \varphi(n)) = 1$. I.e., $\mathsf{pk} = (e, n)$. Note that $\varphi(n) = (p-1)(q-1)$. The secret key is the inverse d of e modulo $\varphi(n)$. Actually, $\mathsf{sk} = (d, n)$ since we still need n to decrypt. To encrypt an element x of $\mathbf{Z}_n$, we compute $y = x^e \bmod n$. To decrypt, we compute $x = y^d \bmod n$. (See Fig. 3.1.)

The key generation takes time $\mathcal{O}(\ell^4)$, where ℓ is the length of the modulus. Encryption and decryption take $\mathcal{O}(\ell^3)$. But if we are up to use a fixed exponent e, the length of e becomes constant and the complexity of encryption becomes $\mathcal{O}(\ell^2)$. One typical example for such e is $e = 2^{16} + 1$. Indeed, raising x to the power e consists of 16 squares and one multiplication. But we shall not forget that p and q must be such that $p-1$ and $q-1$ are coprime with e. Since e is a prime number, this is most likely to be the case!

Compared with the ElGamal cryptosystem, we can see that the complexity of the key generation in RSA is pretty high (if the ElGamal parameters are common to every users, so that they need not to be generated for each of them). But the ElGamal cryptosystem is length-increasing. Finally, one advantage of the ElGamal cryptosystem is that it can adapt to many types of groups (e.g., elliptic curves) whereas it is not so easy to make RSA work outside $\mathbf{Z}_n$.

3.4 Quadratic Residuosity

In a field, the equation $x^2 = 1$ leads to two solutions $+1$ and -1 and nothing more.

Theorem 3.10. *For $x \in K$ where K is a field, we have that $x^2 = 1$ is equivalent to either $x = +1$ or $x = -1$.*

Proof. We have $0 = x^2 - 1 = (x+1)(x-1)$ but in a field, $ab = 0$ is equivalent to either $a = 0$ or $b = 0$. So, $x^2 = 1$ is equivalent to either $x + 1 = 0$ or $x - 1 = 0$. □

Note that in rings, $ab = 0$ does not always imply that $a = 0$ or $b = 0$. For instance, in $\mathbf{Z}_{15}$, we have $3 \times 5 = 0$. We also have $4^2 = 1$. So, we may have more than two roots in rings.

In fields, except when $2 = 0$ (which happens when we work modulo 2), we have $+1 \neq -1$. So, we have exactly two square roots of 1.

The $\mathbf{Z}_p$ case. For p prime and odd, elements of $\mathbf{Z}_p^*$ have either 0 or 2 square roots. Elements with square roots are called *quadratic residues*. The set of all quadratic residues forms a subgroup of $\mathbf{Z}_p^*$. We can easily identify quadratic residues and even extract square roots in this group.

Theorem 3.11. *Given a prime p, an element x of $\mathbf{Z}_p^*$ is a quadratic residue if and only if $x^{\frac{p-1}{2}} \bmod p = 1$.*

Proof. Indeed, if there is a y such that $x = y^2 \bmod p$, we have $x^{\frac{p-1}{2}} \equiv y^{p-1} \equiv 1$. So, a quadratic residue x satisfies $x^{\frac{p-1}{2}} \bmod p = 1$.

Conversely, if $x^{\frac{p-1}{2}} \bmod p = 1$, we can write $x \equiv g^e$ for a generator g (we know that $\mathbf{Z}_p^*$ is cyclic as shown in Th. 2.11). So, $g^{e\frac{p-1}{2}} \equiv 1$. This implies that $p-1$ divides $e\frac{p-1}{2}$, since g is a generator. So, e must be even. We deduce that x is the square of $g^{\frac{e}{2}}$: it is a quadratic residue. □

When $p \equiv 3 \pmod 4$ (so that $\frac{p+1}{4}$ is integer), there is an easy trick to extract square roots which is given by the following result.

Theorem 3.12. *Given a prime p such that $p \equiv 3 \pmod 4$, if $x \in \mathbf{Z}_p^*$ is a quadratic residue, then $x^{\frac{p+1}{4}}$ is a square root of x.*

Proof. First of all, $\frac{p+1}{4}$ is clearly an integer, by the assumption on p. Furthermore, if $x \equiv y^2$, we have $\left(x^{\frac{p+1}{4}}\right)^2 \equiv y^{p+1} \equiv y^2 \equiv x$. So, $x^{\frac{p+1}{4}}$ is a square root of x. □

For any odd prime numbers, square roots can be extracted using the Tonelli algorithm, which works with cubic complexity. The algorithm is as follows.

Input: a prime $p \geq 3$ and a quadratic residue $a \in \mathbf{Z}_p^*$

```
repeat
    choose g ∈ Z_p^* at random
until g is not a quadratic residue
let p − 1 = 2^s t with t odd
e ← 0
for i = 2 to s do
    if (a g^{−e})^{(p−1)/2^i} mod p ≠ 1 then
        e ← 2^{i−1} + e
    end if
end for
output g^{−t e/2} a^{(t+1)/2} mod p
```

Finally, we define the *Legendre symbol*. Given an odd prime number p and an integer x, we define

$$\left(\frac{x}{p}\right) = \begin{cases} 0 & \text{if } x \bmod p = 0 \\ +1 & \text{if } x \text{ is a quadratic residue modulo } p \\ -1 & \text{otherwise} \end{cases}$$

(Note that this is not a fraction: this is a new notation!) Clearly, we have

$$\left(\frac{x}{p}\right) \equiv x^{\frac{p-1}{2}} \pmod p$$

The $\mathbf{Z}_n$ case. In the ring $\mathbf{Z}_n$, quadratic residuosity is more subtle. For instance, for $n = pq$, with two different odd prime numbers p and q, since elements can have two square roots modulo p, two square roots modulo q, we have four combinations which, due to the Chinese Remainder Theorem, reconstruct four different square roots. More precisely, if a is a square root of x modulo p and if b is a square root of x modulo q, then $x \equiv y^2 \pmod n$ is equivalent to $y \bmod p \in \{a, p-a\}$ and $y \bmod q \in \{b, q-b\}$.

We define the *Jacobi symbol*, which generalizes the Legendre symbol: given an odd integer with known factorization $n = p_1^{a_1} \times \cdots \times p_r^{a_r}$, we define

$$\left(\frac{x}{n}\right) = \left(\frac{x}{p_1}\right)^{a_1} \times \cdots \times \left(\frac{x}{p_r}\right)^{a_r}$$

Although $x \in \mathbf{Z}_p^*$ is a quadratic residue if and only if $(x/p) = +1$ (in the odd prime p case), we only have that $x \in \mathbf{Z}_n^*$ is a quadratic residue implies that $(x/n) = +1$ (in the odd composite n case). The converse is not true. There are always some integers x such that $(x/n) = +1$ but which have no square roots. These are "fake" quadratic residues. We can easily construct them by taking non-quadratic residues modulo p and modulo q and combining them using the Chinese Remainder Theorem. Namely, by taking $x_p \in \mathbf{Z}_p$ such that $(x_p/p) = -1$ and $x_q \in \mathbf{Z}_q$ such that $(x_q/q) = -1$, then $x \in \mathbf{Z}_{pq}$ such that $x \bmod p = x_p$ and $x \bmod q = x_q$, we have $(x/p) = (x/q) = -1$ so $(x/pq) = +1$ but x is not a quadratic residue modulo p to it cannot be a quadratic residue modulo pq.

The Jacobi symbol is always easy to compute, even though we may not have the factorization of n at disposal. The algorithm is similar to the Euclid algorithm and is quadratic. Indeed, we have the following properties.

Theorem 3.13. *Let a, b, c be integers.*

- *If b is odd, then $\left(\frac{1}{b}\right) = 1$.*
- *If b is odd, then $\left(\frac{a}{b}\right) = \left(\frac{a \bmod b}{b}\right)$.*
- *If c is odd, then $\left(\frac{ab}{c}\right) = \left(\frac{a}{c}\right)\left(\frac{b}{c}\right)$.*
- *If a is odd, then*
  - *if $a \equiv \pm 1 \pmod 8$, then $\left(\frac{2}{a}\right) = +1$;*
  - *if $a \equiv \pm 3 \pmod 8$, then $\left(\frac{2}{a}\right) = -1$.*
- *If a and b are odd, then*
  - *if $a \equiv b \equiv 3 \pmod 4$, then $\left(\frac{a}{b}\right) = -\left(\frac{b}{a}\right)$;*
  - *otherwise, $\left(\frac{a}{b}\right) = \left(\frac{b}{a}\right)$.*

So, to compute $\left(\frac{a}{b}\right)$ for b odd, we first reduce to $\left(\frac{a \bmod b}{b}\right)$ to have the upper part smaller than the lower part. Then, to compute $\left(\frac{a}{b}\right)$ with $0 \le a < b$, we write $a = 2^\alpha a'$ for some odd a' then use the multiplicativity property to reduce to the computation of $\left(\frac{2}{b}\right)$ (which is covered by the above properties), and of $\left(\frac{a'}{b}\right)$ where a' is odd and lower than b. Then, we use the last property to reduce to the computation of $\left(\frac{b}{a'}\right)$ where the lower part is decreased. We iterate this process and finally obtain a trivial Jacobi symbol to compute.

The mapping $x \mapsto (x/n)$ is a group homomorphism from $\mathbf{Z}_n^*$ to $\{-1, +1\}$. So, the set of all x such that $(x/n) = +1$ is a subgroup of $\mathbf{Z}_n^*$. This subgroup contains another important one: the subgroup QR_n of all quadratic residues. We explain here the properties of QR_n.

- $x \in \mathsf{QR}_n$ implies $(x/n) = +1$. The converse is not true in general. Indeed, there exists non-quadratic residues x which "look like" quadratic residues in the sense that $(x/n) = +1$. These are "fake" quadratic residues.

  When $n = pq$ is the product of two different odd primes p and q, $(x/n) = +1$ is equivalent to $(x/p) = (x/q)$, meaning that the quadratic residuosity status is the same modulo p and modulo q. However, $x \in \mathsf{QR}_n$ is equivalent to $(x/p) = (x/q) = +1$.
- $x \in \mathsf{QR}_n$ and $y \in \mathsf{QR}_n$ imply $xy \in \mathsf{QR}_n$.
- $x \in \mathsf{QR}_n$ and $y \in \mathbf{Z}_n^* - \mathsf{QR}_n$ imply $xy \in \mathbf{Z}_n^* - \mathsf{QR}_n$.
- For p prime, we further have $x \in \mathbf{Z}_p^* - \mathsf{QR}_p$ and $y \in \mathbf{Z}_p^* - \mathsf{QR}_p$ imply $xy \in \mathsf{QR}_p$. (We can see it by using the Legendre symbol of xy.)

  When $n = pq$ is the product of two different odd primes p and q and G is the subgroup of $\mathbf{Z}_n^*$ of all x such that $(x/n) = +1$, for $x \in G$, we have $x \in \mathsf{QR}_n$ is equivalent to $(x/p) = +1$. Hence, $x \in G - \mathsf{QR}_n$ and $y \in G - \mathsf{QR}_n$ imply $xy \in \mathsf{QR}_n$.

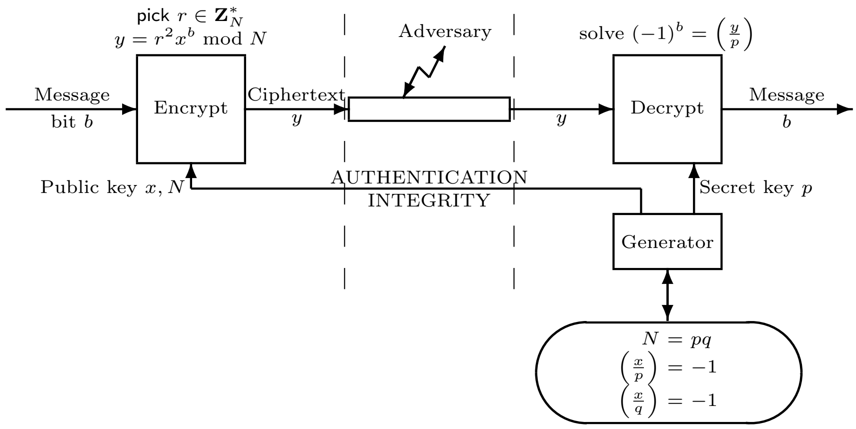

Figure 3.2: The Goldwasser-Micali Cryptosystem

Quadratic residuosity finds many applications in cryptography.

The Goldwasser-Micali cryptosystem. For instance, we can construct the Goldwasser-Micali cryptosystem [41, 42]: we set up the keys by taking $n = pq$ for two different odd primes p and q and $x \in G - \mathsf{QR}_n$. The public key is $\mathsf{pk} = (x, n)$ and the secret key is $\mathsf{sk} = p$. This cryptosystem can encrypt one bit b by $\mathsf{Enc}_{\mathsf{pk}}(b) = r^2 x^b \bmod n$ for $r \in \mathbf{Z}_n^*$ random. We have $(x/p) = -1$. So, we clearly have $(-1)^b = (y/p)$. To decrypt, we can thus define $\mathsf{Dec}_{\mathsf{sk}}(y) = b$ such that $(-1)^b = (y/p)$. (See Fig. 3.2.)

We can also define a primality test. The Solovay-Strassen test [80] checks the primality of an odd p by checking that $b^{\frac{p-1}{2}} = (b/p)$ for a random b. It relies on the following result.

Theorem 3.14. *Let n be an odd number. We have*

$$\begin{array}{lcl} n\ \mathsf{prime} & \Longrightarrow & \forall b \in \mathbf{Z}_n^* \quad b^{\frac{n-1}{2}} \equiv \left(\dfrac{b}{n}\right) \pmod n \\ n\ \mathsf{prime} & \Longleftarrow & \Pr\limits_{b \in \mathbf{Z}_n^*}\left[b^{\frac{n-1}{2}} \equiv \left(\dfrac{b}{n}\right) \pmod n\right] > \dfrac{1}{2} \end{array}$$

for $b \in \mathbf{Z}_n^*$ with uniform distribution.

We can break the DDH problem (seen in the previous chapter) in $\mathbf{Z}_p^*$: let g be a generator of $\mathbf{Z}_p^*$. Given $(X, Y, Z) \in (\mathbf{Z}_p^*)^3$, we let x, y, z be such that $X = g^x$, $Y = g^y$, and $Z = g^z$ in $\mathbf{Z}_p^*$. Then, we can decide if (X, Y, Z) was taken randomly with uniform distribution or if (X, Y) was taken randomly with uniform distribution but z was selected as being $z = xy$, so Z is the solution of the Diffie-Hellman problem in basis g with input (X, Y). For this, we compute $a = 1_{(Z/p)=-1}$, $b = 1_{(X/p)=(Y/p)}$, and compare a and b. If $a = b$ we output 1. Otherwise we output 0.

We first observe that $(g/p) = -1$ because g is a generator. (Otherwise, all group elements would have a Legendre symbol equal to 1, which is incorrect.) So, $(g^z/p) = (-1)^z$. We deduce that $a = 1_{(g^z/p)=-1} = 1_{(-1)^z=-1} = z \bmod 2$. Similarly, we obtain that $b = xy \bmod 2$. Hence, the above attack answer 1 if and only if $xy \equiv z \pmod 2$. If $z = xy$, then the output is 1 for sure. But if z is independent of x and y, then $z = xy$ modulo 2 with probability $\frac{1}{2}$. So, taking a decision by checking $a = b$ gives $\mathsf{Adv} = \frac{1}{2}$, which is much more than deciding at random.

This result implies that the ElGamal cryptosystem is not IND-CPA secure,[1] so unsafe to use in the group $\mathbf{Z}_p^*$. This is why we rather take a subgroup of $\mathbf{Z}_p^*$. But then we have to map messages into this subgroup in an injective and invertible way, and it is not always easy.

[1]The IND-CPA security notion is defined in Def. 7.2. We can show (in another course) that the ElGamal cryptosystem in a group is IND-CPA secure if and only if the DDH problem in this group is hard.

It is easy for the subgroup QR_p of $\mathbf{Z}_p^*$ of quadratic residues modulo p, when $p = 2q + 1$ with p and q prime. In this case, we can see (with the Legendre symbol) that -1 is not a quadratic residue. Indeed, $(-1)^{\frac{p-1}{2}} = (-1)^q = -1$ since q is odd. So, for all $x \in \mathbf{Z}_p^*$, either x or $-x$ is in QR_p but not both. So, we can define $\mathsf{map}(x)$ to be the unique element in $\{x, -x\} \cap \mathsf{QR}_p$. This defines a bijective mapping from $\{1, \ldots, q\}$ to QR_p. So, a message can be first converted into an integer between 1 and q, then into a QR_p element. So, we can use the ElGamal cryptosystem on the QR_p group in this case.

3.5 The Factoring Problem

Given a pseudorandom number generator, the *factoring problem* is specified by a generated integer n. It consists in finding prime factors of n. We define the factoring game relative to algorithm Gen as follows:

$\mathsf{Factoring}(\lambda)$:
1: $\mathsf{Gen}(1^\lambda) \to n$ ▷ λ may represent the modulus bitlength
2: $\mathcal{A}(n) \to (p, q)$
3: **return** $1_{p \times q = n \,\wedge\, p,q \in \{2,\ldots,n-1\}}$

It is actually the problem to split n into non-trivial factor rather than the problem to find the full factorization. For RSA moduli, it is the same.

For instance, we can consider the pseudorandom generator generating RSA moduli $n = pq$. We hope that this problem is hard. So far, the factoring record is obtained on the $\mathsf{RSA768}$ number having 768 bits. We have

$$
\begin{aligned}
&\mathrm{RSA768}\\
&= 123018668453011775513049495838496272077285356959533479219732245215172640050 7\\
&\quad 263657518745202199786469389956474942774063845925192557326303453731548268507 9\\
&\quad 170261221429134616704292143116022212404792747377940806653514195974598569021 4\\
&\quad 3413\\
&= 334780716989568987860441698482126908177047949837137685689124313889828837938 7\\
&\quad 80022876147116525317430877378144679994 89\\
&\quad \times\\
&\quad 367460436667995904282446337996279526322791581643430876426760322838157396665 1\\
&\quad 12792333734171433968102700927987363089 17
\end{aligned}
$$

It was factored by the equivalent of 1500 years of computation on a core 2.2GHz Opteron in 2009. It used NFS, the Number Field Sieve algorithm [56] working with complexity

$$e^{\sqrt[3]{\frac{64}{9}+o(1)}(\ln n)^{\frac{1}{3}}(\ln \ln n)^{\frac{2}{3}}}$$

Another useful factoring algorithm is the Elliptic Curve Method (ECM) which can recover a prime factor p of n with complexity

$$e^{\sqrt{2+o(1)}(\ln p)^{\frac{1}{2}}(\ln \ln p)^{\frac{1}{2}}}$$

which can be better than for NFS if p is small [54]. So, for RSA moduli in which there are two prime factors of equal size, the best algorithm is NFS. But ECM is better for very imbalanced moduli.

With quantum computers, we can factor in quasi-quadratic time using the Shor [78] algorithm. Hence, factoring could become easy if quantum computers with enough memory are built.

Computing square roots relates to factoring as the following result shows.

Theorem 3.15. *We consider a pseudorandom generator* Gen *to produce an RSA modulus* n *of size* ℓ. *We consider the two following problems:*

- *find the two factors* p *and* q *of* n*;*

- *given a random quadratic residue $x \in \mathbf{Z}_n$, finds a square root y of x.*

Any algorithm solving one of the two problem can be transformed into an algorithm solving the other problem, with a complexity which is polynomially bounded in terms of ℓ.

Proof (sketch). Clearly, being able to factor implies being able to compute square roots: we just have to compute square roots modulo each factor p and q of n using the Tonelli algorithm and to combine them with the Chinese Remainder Theorem. This solves the square root problem in $\mathbf{Z}_n$.

Conversely, if we can compute square roots, by computing the square root of $x^2 \bmod n$, for some random x, it is likely to give a different square root than x or $-x$. If we have $x^2 \equiv y^2$ with $x \neq y$ and $x \neq -y$, the result below yields a non-trivial factor of n, so p or q. Then, we can fully factor n. $\square$

Lemma 3.16. *For $x, y \in \mathbf{Z}_n$*

$$\left.\begin{array}{rcl} x^2 & \equiv & y^2 \pmod n \\ x & \not\equiv & y \pmod n \\ x & \not\equiv & -y \pmod n \end{array}\right\} \Longrightarrow \mathsf{gcd}(x-y, n) \notin \{1, n\}$$

Proof. If we have $x^2 \equiv y^2$ with $x \neq y$ and $x \neq -y$, we obtain that $(x-y)(x+y)$ is a multiple of n but that n is not a factor of either $x-y$ or $x+y$. So, $\mathsf{gcd}(x-y, n)$ is a non-trivial factor of n. For $n = pq$, this is either p or q. $\square$

There are other problems related to the factoring problem. For instance, if we know $\lambda(n)$ or any multiple (such as $\varphi(n)$), we can factor n by writing $\lambda(n) = 2^s t$ with t odd, computing the chain $(b^t \bmod n, b^{2t} \bmod n, \ldots, b^{2^s t} \bmod n)$ for a random b. If the chain is of the form $(1, 1, \ldots, 1)$ or $(*, \ldots, *, -1, 1, \ldots, 1)$, this is bad luck and we can try with another b. Otherwise, we obtain some z such that $z \neq 1$, $z \neq -1$, but $z^2 \bmod n = 1$. So, $\mathsf{gcd}(z-1, n)$ is a non-trivial factor of n.

To conclude, we list some computational problems related to $\mathbf{Z}_n$ which are equivalent to factoring n:

- computing square roots in $\mathbf{Z}_n$;
- computing $\varphi(n)$;
- computing $\lambda(n)$;
- computing a multiple of $\lambda(n)$.

We can deduce, for instance, that computing an RSA secret key (d, n) from the public key (e, n) is as hard as factoring n since $ed - 1$ must be a multiple of $\lambda(n)$. This means that the RSA key recovery problem is as hard as the factoring problem.

Regarding the RSA decryption problem, it can of course be solved if factoring is easy but the two problems are not known to be equivalent. It is believed to be hard, but maybe not equivalent.

Chapter 4

Elliptic Curve Cryptography

This chapter provides basic facts about elliptic curves over finite fields and how to use them in cryptography.

4.1 Galois Fields

It can be proven that all finite fields have a cardinality of form p^k where p is a prime number. This prime number p is called the *characteristic* of the field. Moreover, finite fields of the same cardinality must be isomorphic. Conversely, for any prime number p and any positive integer k, we can construct a finite field, denoted by $\mathsf{GF}(p^k)$, with a cardinality of p^k. For that, we can just construct the quotient of the ring $\mathbf{Z}_p[X]$ of polynomials with coefficients in $\mathbf{Z}_p$ by the ideal generated by a monic irreducible polynomial $P(X)$ of degree k. (Such polynomial always exists!) So, field elements can be seen as polynomials with coefficient in $\mathbf{Z}_p$ and degree at most $k-1$. Operations are addition and multiplication of polynomials modulo $P(X)$ and modulo p.

In cryptography, we are mostly interested in fields with cardinality p for a large prime p, i.e., the field $\mathbf{Z}_p$, and by fields of cardinality 2^k for some integer k. The former is called a *prime field*. The latter is called a *binary field*. Indeed, it consists of polynomials with binary coefficients and degree at most $k-1$. Operations are taken modulo a fixed polynomial $P(X)$ and coefficients are reduced modulo 2. Note that the *representation* of binary field implies the selection of this reference $P(X)$. So, binary fields can have different (isomorphic) representations.

In the AES block cipher (to be seen in another chapter), we use the field $\mathsf{GF}(2^8)$ with $P(X) = X^8 + X^4 + X^3 + X + 1$. Since elements are polynomials with binary coefficient and degree up to 7, they are represented as bytes. If $a = a_7X^7 + \cdots + a_1X + a_0$, it is represented by the bitstring $a_7 \cdots a_1a_0$. Clearly, the addition in the field corresponds to the XOR: the bitwise exclusive OR. Multiplication by the byte `0x01` is trivial: $a \times \texttt{0x01} = a$. We detail now what is the multiplication by the byte `0x02`. Since this byte represents the polynomial X, to multiply by X we just shift by one bit to the left. Now, if shifting makes the monomial X^8 appear, we can reduce it modulo $P(X)$, which means that we drop X^8 and add (i.e., XOR) the polynomial $X^4 + X^3 + X + 1$ which is represented by `0x1b`. So, when implementing the multiplication by `0x02`, we just shift the byte a by one bit to the left and XOR it to `0x1b` if there is a carry bit. I.e., if the byte had a most significant bit of 1 before shifting. Finally, to multiply by `0x03`, we can just multiply by `0x01` and by `0x02` and add (i.e., XOR) the two results. In AES, we only need to multiply by `0x01`, `0x02`, and `0x03`.

In binary fields, we note that $-a = a$ for all a. We also have $(a + b)^2 = a^2 + b^2$ for all a and b: the square operation is a linear one! So, raising to the power 2^i is also linear. Square roots are unique: to compute the square root of a, we can just take $a^{2^{k-1}}$. (Indeed, $a^{2^k} = a$ in a field of 2^k elements.) Finally, there is a useful function called *trace* on $\mathsf{GF}(2^k)$. We define

$$\mathsf{Tr}(a) = a + a^2 + a^{2^2} + \cdots + a^{2^{k-1}}$$

Due to the linearity of the square, this is a linear function. Due to the fact that $a^{2^k} = a$, we can see that

$$\begin{aligned}\mathsf{Tr}(a)^2 &= (a + a^2 + a^{2^2} + \cdots + a^{2^{k-1}})^2 \\ &= a^2 + a^{2^2} + a^{2^3} + \cdots + a^{2^{k-1}} + a^{2^k} \\ &= a^2 + a^{2^2} + a^{2^3} + \cdots + a^{2^{k-1}} + a \\ &= \mathsf{Tr}(a)\end{aligned}$$

So, $\mathsf{Tr}(a)^2 = \mathsf{Tr}(a)$. This implies that $\mathsf{Tr}(a)$ is a root of $x^2 = x$, i.e., that $x(x-1) = 0$. Hence, $\mathsf{Tr}(a) \in \{0, 1\}$. Therefore, the trace is a linear function from $\mathsf{GF}(2^k)$ to $\mathbf{Z}_2$. Furthermore, $\mathsf{Tr}(x^2) = \mathsf{Tr}(x)$.

Here is an example of application of the trace function which will be used later.

Theorem 4.1. *In $\mathsf{GF}(2^k)$, the equation $a = x^2 + x$ in x has solutions if and only if $\mathsf{Tr}(a) = 0$. When there are solutions, there are exactly two which can be written $x = \theta$ and $x = \theta + 1$ for some θ.*

Proof. Since we have an equation of degree 2, we cannot have more than two solutions. Indeed, if θ is a solution, then $a = x^2 + x$ implies $0 = x^2 - \theta^2 + x - \theta$. So, $0 = (x - \theta)(x - \theta + 1)$. We deduce that either $x = \theta$ or $x = \theta + 1$. In any case, there are no more than 2 solutions.

Since $\mathsf{Tr}(x)^2 = \mathsf{Tr}(x)$ and since the trace is linear, when there is a solution x, we must have

$$\mathsf{Tr}(a) = \mathsf{Tr}(x^2 + x) = \mathsf{Tr}(x^2) + \mathsf{Tr}(x) = \mathsf{Tr}(x) + \mathsf{Tr}(x) = 0$$

To show the converse (i.e. that $\mathsf{Tr}(a) = 0$ implies the existence of a solution), we observe that the mapping $x \mapsto x^2 + x$ in $\mathsf{GF}(2^k)$ has preimage sets of size limited to two, due to the above observation. It maps to elements of trace 0, i.e. to half of the $\mathsf{GF}(2^k)$ set. So, there are at most 2^{k-1} preimage sets, each of them with at most 2 elements. Since this must cover the entire 2^k elements, we deduce that exactly all the elements of trace 0 have a non-empty preimage set and that all non-empty preimage sets have cardinality exactly 2. Hence, an element of trace 0 has always 2 preimages. □

4.2 Elliptic Curves

Over the field $\mathbf{R}$, an elliptic curve with parameters a and b consists of a special point $\mathcal{O}$ called the *point at infinity* and of the points (x, y) which are solutions of the equation $y^2 = x^3 + ax + b$. Since (x, y) being a solution to this equation implies that $(x, -y)$ is also a solution to this equation, the curve is symmetric by the x axis. Since the equation is cubic, the chord passing through two points $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$ of the curve normally intersects the curve on a third point. When this is the case, we let $R = (x_R, y_R)$ be the symmetric of this third point and call it $R = P + Q$. (See Fig. 4.1.) There are cases when some of these three points may be equal, or when some of them are the points at infinity. We discuss all cases below.

More precisely, by writing the slope of the chord

$$\lambda = \frac{y_Q - y_P}{x_Q - x_P}$$

we intersect $y = \lambda x + \mu$ (for some μ) with $y^2 = x^3 + ax + b$. By substituting y we have $x^3 - (\lambda x + \mu)^2 + ax + b = 0$. We know that x_P and x_Q are roots of this equation. The sum of the three roots of this equation is the opposite of the x^2 coefficient, i.e., λ^2. So, the sum of the x coordinates of the three points of intersection is λ^2. Hence, $x_R = \lambda^2 - x_P - x_Q$. We can further obtain $y_R = (x_P - x_R)\lambda - y_P$. We can thus compute $P + Q$ analytically.

When P and Q are symmetric of each other, we have $x_P = x_Q$ and the slope is infinite. In this case, we say $P + Q = \mathcal{O}$, the point at infinity. Geometrically, we say that $\mathcal{O}$ is intersecting the

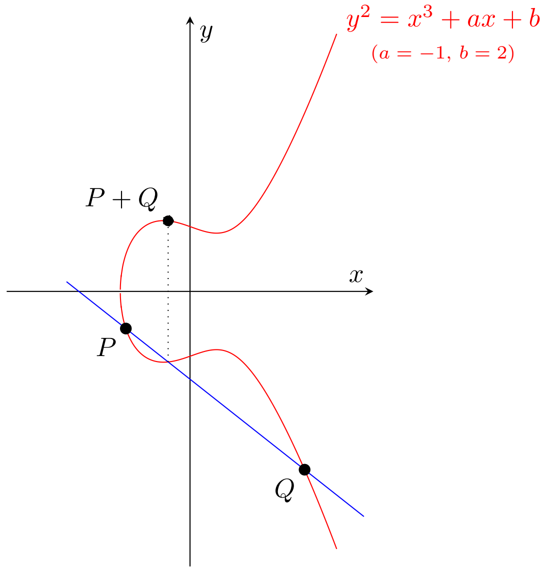

Figure 4.1: An Elliptic Curve

vertical chord and the curve at infinity. I.e., $\mathcal{O}$ is the third point of intersection. By convention, the symmetric of $\mathcal{O}$ is $\mathcal{O}$ itself. If Q is the symmetric of P, we write $Q = -P$ since $P + Q = \mathcal{O}$.

Conversely, we can also consider the chord between P and $Q = \mathcal{O}$. By convention, this is the vertical line going through P. So, the third intersection point is $-P$, and its symmetric is $R = P$. We have $P + \mathcal{O} = P$. So, we define $\mathcal{O}$ to be the neutral element.

In the $P = Q$ case, we replace the chord by the tangent (which is a line with a double contact point to the curve). But for the curve to have a tangent on any point, there should not be any *singular point*. I.e., we need the curve to be *non-singular*. To obtain the slope λ, we say that x_P must be a double-root of $x^3 - (\lambda x + \mu)^2 + ax + b = 0$. I.e., x_P must be a root of the derivative $3x^2 - 2\lambda(\lambda x + \mu) + a = 0$. Since $y_P = \lambda x_P + \mu$, we just have $3x_P^2 - 2\lambda y_P + a = 0$ from which we deduce

$$\lambda = \frac{3x_P^2 + a}{2y_P}$$

This is for $y_P \neq 0$. For $y_P = 0$, P is self-inverse, so $P + P = \mathcal{O}$ in this case.

To make sure that all points of the curve have a tangent, we must make sure that the differential of $f(x, y) = y^2 - (x^3 + ax + b)$ on any point is not zero. This means that either $\frac{\partial f}{\partial x} \neq 0$ or $\frac{\partial f}{\partial y} \neq 0$. This means that either $-(3x^2+a) \neq 0$ or $2y \neq 0$ on any point (x, y) which is solution to $f(x, y) = 0$. So, this condition is equivalent to $3x^2 + a \neq 0$ on all x such that $x^3 + ax + b = 0$. This is equivalent to the polynomials $x^3 + ax + b$ and $3x^2 + a$ having no common root. A known criterion for this is that the *resultant* of the two polynomials is nonzero. The resultant can be computed as the determinant of the *Sylvester matrix* of the two polynomials, i.e.

$$\mathsf{Res} = \begin{vmatrix} 1 & 0 & a & b & 0 \\ 0 & 1 & 0 & a & b \\ 3 & 0 & a & 0 & 0 \\ 0 & 3 & 0 & a & 0 \\ 0 & 0 & 3 & 0 & a \end{vmatrix} = 4a^3 + 27b^2$$

(We can see that if the polynomials have some common root x, then the matrix multiplied by the column $(x^4, x^3, x^2, x, 1)^t$ vanishes. Conversely, if the determinant vanishes, then there must be a linear combination of the rows which is zero, which expresses as the existence of some $(u_1, u_2, u_3, u_4, u_5)$ such that $(x^3 + ax + b)(u_1x + u_2) + (3x^2 + a)(u_3x^2 + u_4x + u_5) = 0$. By factoring

$3x^2 + a$ we can see that this implies that the gcd between $x^3 + ax + b$ and $3x^2 + a$ has degree at least 1, so these two polynomials have a root in common, at least in an extension field.) So, the curve (in any extension field) has a tangent to any point if and only if $4a^3 + 27b^2 \neq 0$.

It is not easy to see that the addition defines an Abelian group structure on the curve, but it does.

The most frequent usage of elliptic curve in cryptography assumes that the discrete logarithm problem is hard. So, it is used as an alternate structure as $\mathbf{Z}_p^*$. Contrarily to $\mathbf{Z}_p^*$ which is always cyclic, elliptic curves are not always cyclic. This can be easily seen by looking at the number of elements of order 2 in the group. In a cyclic group, there can only be one element of order 2, at most. In the above elliptic curve, a point P has order 2 if and only if $P = -P$ and $P \neq \mathcal{O}$. This is equivalent to $P = (x, y)$ such that $2y = 0$ and $x^3 + ax + b = y^2$. Except when the field has a characteristic two, this is equivalent to $y = 0$ and $x^3 + ax + b = 0$. So, the number of points of order 2 is equal to the number of roots of the polynomial $x^3 + ax + b$. When it has more than one root, the elliptic curve cannot be cyclic.

In the next sections, we study more the elliptic curves over finite fields.

4.3 Elliptic Curves over a Prime Field

This section presents the case of elliptic curves over $\mathbf{Z}_p$, for $p > 3$ prime. (Or other fields of characteristic p, actually.) Note that the $p = 3$ case is special and will not be covered in this document.

Given the field $\mathbf{K}$ of characteristic $p > 3$ and two field elements a and b, we define

$$E_{a,b}(\mathbf{K}) = \{\mathcal{O}\} \cup \{(x, y) \in \mathbf{K}^2; y^2 = x^3 + ax + b\}$$

The *discriminant* is $\Delta = -16(4a^3 + 27b^2)$. The curve is *non-singular* if an only if $\Delta \neq 0$, and we only cover this case below. We define the point addition over $E_{a,b}(\mathbf{K})$ to give it an Abelian group structure:

- for $P = (x_P, y_P)$, we let $-P = (x_P, -y_P)$ and $-\mathcal{O} = \mathcal{O}$;
- for $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$, if $Q = -P$ we let $P + Q = \mathcal{O}$;
- for $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$, if $Q \neq -P$ we let

$$\begin{aligned} \lambda &= \begin{cases} \frac{y_Q - y_P}{x_Q - x_P} & \text{if } x_P \neq x_Q \\ \frac{3x_P^2 + a}{2y_P} & \text{if } x_P = x_Q \end{cases} \\ x_R &= \lambda^2 - x_P - x_Q \\ y_R &= (x_P - x_R)\lambda - y_P \end{aligned}$$

  and $P + Q = R$, where $R = (x_R, y_R)$;
- for the addition to $\mathcal{O}$, we let $P + \mathcal{O} = \mathcal{O} + P = P$ and $\mathcal{O} + \mathcal{O} = \mathcal{O}$.

Given $u \in \mathbf{K}$, we can easily see that the mapping f from $E_{a,b}(\mathbf{K})$ to $E_{u^4a,u^6b}(\mathbf{K})$ defined by $f(x, y) = (u^2x, u^3y)$ and $f(\mathcal{O}) = \mathcal{O}$ is a *group isomorphism*. So, given $v \in \mathbf{K}$, the two curves $E_{a,b}(\mathbf{K})$ and $E_{v^2a,v^3b}(\mathbf{K})$ are isomorphic when v is a quadratic residue (indeed, we can write $v = u^2$ for some u in this case). When v is not a quadratic residue, we say that the two curves are *twist* of each other. We note that in this case, the curves are isomorphic when considered as curves in a field extension of $\mathbf{K}$ (namely, in a super-field in which v becomes a quadratic residue), although they are not isomorphic when considered as curves over $\mathbf{K}$.

We define the *j-invariant* $j = 1728\frac{4a^3}{4a^3+27b^2}$. We can prove that two isomorphic curves have the same j-invariant. (We can easily see this on the above example.) The j-invariant is actually a characteristic of classes of isomorphic curves, as the following result shows.

Theorem 4.2. *Over a field* $\mathbf{K}$*, isomorphic curves have the same* j*-invariant. Conversely, curves with the same* j*-invariant are either isomorphic or twist of each other.*

So, curves are isomorphic in $\mathbf{K}$ or one of its extensions if and only if they have the same j-invariant.

Finally, the order of the curve is close to $\#\mathbf{K}$. This is due to $x^3 + ax + b$ of having two square roots in nearly half of the cases. It is usually written $\#E_{a,b}(\mathbf{K}) = \#\mathbf{K} + 1 - t$ where t is called the *trace of Frobenius*. (So, t is small.) We have a more precise result below.

Theorem 4.3 (Hasse Theorem). *Over a field* $\mathbf{K}$*, the trace of Frobenius* t *of an elliptic curve satisfies* $|t| \leq 2\sqrt{\#\mathbf{K}}$.

We can easily show that two twisted curves such as $E_{a,b}(\mathbf{K})$ and $E_{v^2a,v^3b}(\mathbf{K})$ for $v \in \mathbf{K}$ which is not a square have a cumulated cardinality of $2 + 2\#\mathbf{K}$. So, the trace of Frobenius is opposite. Indeed, there are the two neutral elements. Then, for each x, the number of $(x, y) \in E_{a,b}(\mathbf{K})$ plus the number of $(vx, y') \in E_{v^2a,v^3b}(\mathbf{K})$ is always equal to 2: if $x^3 + ax + b = 0$, then both are equal to 1. Otherwise one of the two is equal to 2 and the other is equal to 0. Indeed, $x^3 + ax + b$ is a nonzero square if and only if $(vx)^3 + (v^2a)(vx) + (v^3b)$ is a nonzero non-square. Since $\mathbf{K}$ is a finite field with characteristic different from 2, each nonzero square has exactly two square roots.

4.4 Elliptic Curve over a Binary Field

Before looking in more details at elliptic curves over binary field, we should warn that recent cryptanalytic results from Joux and Vitse [48] raise concerns about the hardness of the discrete logarithm over these curves. So, many are insecure to use. However, on $\mathsf{GF}(2^k)$ with k prime to avoid subfields, the security is still open.

Over a field $\mathbf{K}$ of characteristic 2, except for the special case of *supersingular curves* which will be treated below, *ordinary* curves are defined by two field elements denoted by a_2 and a_6. We define

$$E_{a_2,a_6}(\mathbf{K}) = \{\mathcal{O}\} \cup \{(x, y) \in \mathbf{K}^2; y^2 + xy = x^3 + a_2x^2 + a_6\}$$

The *discriminant* is $\Delta = a_6$. The curve is *non-singular* if and only if $\Delta \neq 0$, and we only cover this case below. We define a point addition over $E_{a_2,a_6}(\mathbf{K})$ to give it an Abelian group structure:

- for $P = (x_P, y_P)$, we let $-P = (x_P, x_P + y_P)$ and $-\mathcal{O} = \mathcal{O}$;
- for $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$, if $Q = -P$ we let $P + Q = \mathcal{O}$;
- for $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$, if $Q \neq -P$ we let

$$\begin{aligned} \lambda &= \begin{cases} \frac{y_Q+y_P}{x_Q+x_P} & \text{if } x_P \neq x_Q \\ \frac{x_P^2+y_P}{x_P} & \text{if } x_P = x_Q \end{cases} \\ x_R &= \lambda^2 + \lambda + a_2 + x_P + x_Q \\ y_R &= (x_P + x_R)\lambda + y_P + x_R \end{aligned}$$

  and $P + Q = R$ where $R = (x_R, y_R)$;
- for the addition to $\mathcal{O}$, we let $P + \mathcal{O} = \mathcal{O} + P = P$ and $\mathcal{O} + \mathcal{O} = \mathcal{O}$.

Given $u \in \mathbf{K}$, we can easily see that the mapping f from $E_{a_2,a_6}(\mathbf{K})$ to $E_{a_2+u^2+u,a_6}(\mathbf{K})$ defined by $f(x, y) = (x, ux + y)$ and $f(\mathcal{O}) = \mathcal{O}$ is a *group isomorphism*. So, given $v \in \mathbf{K}$, the two curves $E_{a_2,a_6}(\mathbf{K})$ and $E_{a_2+v,a_6}(\mathbf{K})$ are isomorphic when $\mathsf{Tr}(v) = 0$ (indeed, we can write $v = u^2 + u$ for some u in this case). When $\mathsf{Tr}(v) = 1$, they are only isomorphic as curves in a field extension of $\mathbf{K}$ (namely, an extension in which the trace of v vanishes), but not as curves over $\mathbf{K}$. In this case, we say that the two curves are *twist* of each other.

We define the *j-invariant* $j = 1/\Delta$. Like for the prime field case, we can prove that two isomorphic curves have the same j-invariant (this can be seen on the above example). Conversely, two curves with same j-invariant are again isomorphic (as curves) in some extension field. Over the field $\mathbf{K}$, they are either isomorphic or twist of each other.

Finally, the order of the curve is close to $\#\mathbf{K}$. This is due to $y^2+xy = x^3+a_2x^2+a_6$ of having two solutions y for nearly half of the x. It is usually written $\#E_{a_2,a_6}(\mathbf{K}) = \#\mathbf{K}+1-t$ where t is called the *trace of Frobenius*. (So, t is small.) More precisely, we know that $|t| \leq 2\sqrt{\#\mathbf{K}}$. This is the Hasse Theorem.

Supersingular curves. There exists a special type of curves over binary fields which are called *supersingular curves*. (This notion should not be confused with the notion of singular curves.) They could be used to construct pairings as we will study later in this chapter.

For supersingular curves, three field elements denoted by a_3, a_4, and a_6 define

$$E_{a_3,a_4,a_6}(\mathbf{K}) = \{\mathcal{O}\} \cup \{(x,y) \in \mathbf{K}^2; y^2 + a_3y = x^3 + a_4x + a_6\}$$

The *discriminant* is $\Delta = a_3^4$. The curve is *non-singular* if an only if $\Delta \neq 0$, and we only cover this case below. We define a point addition over $E_{a_3,a_4,a_6}(\mathbf{K})$ to give it an Abelian group structure:

- for $P = (x_P, y_P)$, we let $-P = (x_P, y_P + a_3)$ and $-\mathcal{O} = \mathcal{O}$;
- for $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$, if $Q = -P$ we let $P + Q = \mathcal{O}$;
- for $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$, if $Q \neq -P$ we let

$$\begin{aligned}
\lambda &= \begin{cases} \frac{y_Q+y_P}{x_Q+x_P} & \text{if } x_P \neq x_Q \\ \frac{x_P^2+a_4}{a_3} & \text{if } x_P = x_Q \end{cases} \\
x_R &= \lambda^2 + \lambda + x_P + x_Q \\
y_R &= (x_P + x_R)\lambda + y_P + a_3
\end{aligned}$$

  and $P + Q = R$ where $R = (x_R, y_R)$;
- for the addition to $\mathcal{O}$, we let $P + \mathcal{O} = \mathcal{O} + P = P$ and $\mathcal{O} + \mathcal{O} = \mathcal{O}$.

The *j-invariant* is $j = 0$, and all these curves are isomorphic.

4.5 Elliptic Curve Factoring

One simple factoring method (which is not so good) is Pollard's so-called $p-1$ algorithm. It works as follows:

Input: n s.t. the largest prime factor of $p-1$ is at most B
Output: a nontrivial factor of n
Complexity: $\mathcal{O}(B)$ arithmetic operations

pick x at random in $\{2, \ldots, n-1\}$
if $\mathsf{gcd}(x, n) \neq 1$ **then**
    output this gcd and stop
end if
$i \leftarrow 1$
while $\mathsf{gcd}(x-1, n) = 1$ **do**
    $x \leftarrow x^i \bmod n$
    $i \leftarrow i + 1$
end while
if $x = 1$ **then**
    fail
else
    output $\mathsf{gcd}(x-1, n)$ and stop

end if

When $n = pq$, we assume that the largest prime factor B of $p-1$ is smaller than the largest prime factor of $q-1$. This way, $x^{B!} \bmod p$ should become 1 with good chances while $x^{B!} \bmod q > 1$. Hence, $\mathsf{gcd}((x^{B!}-1) \bmod n, n) = p$. The above algorithm iteratively compute $x^{i!} \bmod n$ and checks if $\mathsf{gcd}((x^{i!}-1) \bmod n, n) \neq 1$. This works pretty well when one factor p of n is such that $p-1$ is smooth. However, we have troubles when

$$\max_{r|p-1 \text{ prime}} r = \max_{r|q-1 \text{ prime}} r$$

as factorial numbers are likely to be multiples of $p-1$ and $q-1$ simultaneously, which would make the $p-1$ method fails almost all the time.

We can adapt the above algorithm by replacing the $\mathbf{Z}_p^*$ group by an elliptic curve over $\mathbf{Z}_p$. The magic is that we can define such curve without knowing p and do nearly any computations over it. We select the elliptic curve randomly and use the probability that such elliptic curve will have a smooth degree. We obtain this way the *Elliptic Curve Method (ECM)*. It works with complexity

$$\mathcal{O}\left(e^{\sqrt{(1+o(1))\log p \log\log p}}\right)$$

where p is the smallest prime factor of n. It is the best method to find p when it is small. The algorithm is given below:

Input: n
Output: a nontrivial factor of n
Complexity: $\mathcal{O}(B)$ arithmetic operations where

$$B \approx \min_{p \text{ prime factor of } n} E_{N\in[p-2\sqrt{p},p+2\sqrt{p}]}\left(\max_{q \text{ prime factor of } N} q\right)$$

pick a and $X = (x, y)$ at random in $\mathbf{Z}_n$
let b such that $y^2 \equiv x^3 + ax + b \pmod n$
$i \leftarrow 1$
repeat
    $i \leftarrow i + 1$
    $X \leftarrow i \cdot X$ over the curve (modulo n)
until division error modulo n
if divisor multiple of n then fail
output $\mathsf{gcd}(\mathsf{divisor}, n)$

Essentially, we take a random point $X = (x, y)$ then a random elliptic curve going through this point. This means we select a at random and fix b such that $y^2 = x^3 + ax + b$ modulo n (as this would be true modulo p as well). Then we compute iteratively $i!.X$. We use the standard double-and-add algorithm with point addition. To compute point addition, we do the operations modulo n (as they would be true modulo p as well). We just have a problem to check equalities modulo p. But we only need it to rule out the $P + (-P)$ computation which gives the point at infinity. For this, we omit the verification and proceed with the standard addition rule. If we ever try to add P and $-P$, we should have a division by zero modulo p. This would translate into a division by a multiple of p modulo n. This would be non-invertible. This typically happens precisely when $i!.X$ becomes the point at infinity. Then, the multiple of p leaks p by computing a gcd. It is a case where an exception in the computation gives the precise result we were looking for!

4.6 Using Elliptic Curves

Elliptic curves are often used in cryptography. In most cases, we use elliptic curves in which the discrete logarithm problem is hard. We list below a few facts about the discrete logarithm problem.

- The discrete logarithm is very easy to compute in *anomalous* elliptic curves over $\mathbf{Z}_p$ (these are curves of order p).
- Binary curves may be exposed to recent attacks [48].
- There are some other families of elliptic curves in which the discrete logarithm is easy to compute.
- In a cyclic group of order n, the Pollard Rho algorithm [64] solves the discrete logarithm problem in complexity $\mathcal{O}(\sqrt{n})$.
- We can consider tradeoffs. For instance, Bernstein and Lange [17] propose to run a precomputation of complexity $\mathcal{O}(n^{\frac{2}{3}})$ for a curve then compute any discrete logarithm in the curve with complexity $\mathcal{O}(n^{\frac{1}{3}})$. As people tend to use the very same curves, this may be a devastating attack.

The choice of a curve must be specified in the *domain parameters*. Typically, this includes

- the choice for a finite field $\mathbf{K}$, so, either some large prime p (for $\mathbf{K} = \mathbf{Z}_p$), or some $q = 2^k$ (for $\mathsf{GF}(2^k)$) together with the selection of an irreducible binary polynomial $P(X)$ of degree k;
- the field elements defining the curve E over $\mathbf{K}$;
- a point $G \in E$ and its order n in the group;
- sometimes, the *cofactor* h of the group, i.e. $h = \frac{\#E}{n}$;
- sometimes, as well, the seed which was used to generate the above parameters.

In the last option, specifying the seed can be used to convince that the curve was not maliciously generated.

As we can see, we must manipulate modulo p integers, polynomials, field elements, elliptic curve points, in addition to bitstrings and bytes. To convert from one type to the other, it is important to have standards. We refer to `http://www.secg.org/download/aid-385/sec1_final.pdf` for this. Points are represented as byte strings. The first byte is special. If it is `0x00`, the string has only one byte and this represents the point at infinity $\mathcal{O}$. If it is `0x04`, what follows is the encoding of x followed by the encoding of y. If the first byte is either `0x02` or `0x03`, what follows is only the encoding of x. This is the *point compression* trick.

Indeed, for *prime field curves*, we know that y is a solution of

$$y^2 = x^3 + ax + b$$

There are two solutions which are opposite of each other modulo p. So, they do not have the same parity. The point we refer to is the one in which the parity of y is the same as the parity of this special byte.

For *ordinary curves over binary fields*, we know that $\frac{y}{x}$ is a solution of

$$\left(\frac{y}{x}\right)^2 + \frac{y}{x} = x + a_2 + \frac{a_6}{x^2}$$

The two solutions have a sum equal to 1, so have different traces. The point we refer to is the one in which the trace of $\frac{y}{x}$ is the parity of this special byte.

There are several standards for elliptic curves: FIPS 186-2 by NIST, SEC2 by the SECG consortium, X9.62 by ANSI, and IEEE P1363. They propose many curves in common, but under different names. The NIST standard includes pseudorandom curves over $\mathbf{Z}_p$ such as $\mathsf{P192}$ or $\mathsf{P256}$, and ordinary curves over binary fields. This includes pseudorandom curves such as $\mathsf{B163}$, and a special type of curve called *Koblitz curves*, or *anomalous binary curves*. The SECG standard propose the same type of curve and also another special type of curve over prime fields called *generalized Koblitz curves*. The curve P192 exists under the name secp192r1. The curve P256 exists under the name secp256r1. The curve B163 exists under the name sect163r2. For instance, the curve P256 is the one used in the Swiss biometric passports[1].

Here is the curve secp192r1 = P192:

$$\mathsf{secp192r1} = \{\mathcal{O}\} \cup \{(x,y) \in \mathbf{Z}_p; y^2 = x^3 + ax + b\}$$

$$\begin{aligned}
p &= 6277101735386680763835789423207666416083908700390324961279\\
a &= p - 3\\
b &= 2455155546008943817740293915197451784769108058161191238065\\
n &= 6277101735386680763835789423176059013767194773182842284081\\
G &= \texttt{03 188da80e b03090f6 7cbf20eb 43a18800 f4ff0afd 82ff1012}\\
&= 03 : 602046282375688656758213480587526111916698976636884684818\\
\mathsf{seed} &= \texttt{3045ae6f c8422f64 ed579528 d38120ea e12196d5}
\end{aligned}$$

note that $p = 2^{192} - 2^{64} - 1$, $2^{192} - 2^{95} < n < 2^{192}$, and n is prime.

Here is the curve sect163r2 = B163:

$$\mathsf{sect163r2} = \{\mathcal{O}\} \cup \{(x,y) \in \mathsf{GF}(q); y^2 + xy = x^3 + a_2x^2 + a_6\}$$

$$\begin{aligned}
q &= 2^{163}\\
P(x) &= x^{163} + x^7 + x^6 + x^3 + 1\\
a_2 &= 1\\
a_6 &= \texttt{02 0a601907 b8c953ca 1481eb10 512f7874 4a3205fd}\\
n &= 5846006549323611672814742442876390689256843201587\\
G &= \texttt{03 03 f0eba162 86a2d57e a0991168 d4994637 e8343e36}\\
\mathsf{seed} &= \texttt{85e25bfe 5c86226c db12016f 7553f9d0 e693a268}
\end{aligned}$$

note that $2^{162} < n < 2^{162} + 2^{82}$ and n is prime.

X25519 is another elliptic curve which has become very popular, specially after a scandal that the NSA influenced standardization to adopt a curve in which they knew a trapdoor, because it was not proposed by any company or government agency and because it had very efficient implementations. The curve Curve25519 was proposed in 2005 by Bernstein [18]. Since then, it was adopted in SSH, Tor, Signal (the secure messaging protocol used in WhatsApp, discussed in Section A.4), Bitcoin (discussed in Section A.2), etc. We have

$$\mathsf{Curve25519} = \{\mathcal{O}\} \cup \{(x,y) \in \mathbf{Z}_p; y^2 = x^3 + 486\,662x^2 + x\}$$

(note that the equation has a different form than the one we used before) with $p = 2^{255} - 19$ (which is prime). A proposed base point G has the x-coordinate $x_G = 9$. It has order

$$2^{252} + 27742317777372353535851937790883648493$$

(which is also prime). In addition to this, a function X25519 is used for the Elliptic Curve Diffie-Hellman protocol (ECDH).

4.7 Elliptic Curve Cryptography

The ECDH key exchange protocol is the variant of the Diffie-Hellman protocol working over an elliptic curve as a group. It is specified in the SEC1 standard on `http://www.secg.org/`

[1] See Section A.8.

`collateral/sec1.pdf`, and also in IEEE P1363. It is used in Bluetooth 2.1[2] and in the European biometric passports (in the Extended Access Control protocol, see Section A.8). We have two participants U and V using the same subgroup of order n generated by some point G over an elliptic curve. (Let assume that the group has order hn and that n is prime.) They both select their secret key $d_U, d_V \in \mathbf{Z}_n^*$, respectively. They compute their public keys $Q_U = d_U.G$ and $Q_V = d_V.G$, which are points, and exchange them. Then, they both check that the received public key is actually a point of the curve which is generated by G (see below), different from the point at infinity, and that its order is a factor of n. They both compute a point P, either by $P = d_U.Q_V$ or by $P = d_V.Q_U$. They take the first coordinate x_P of P and convert it into a byte string Z. Finally, they compute $K = \mathsf{KDF}(Z)$ by using a *Key Derivation Function* KDF. (This notion will be covered in another chapter.)

In the security of ECDH, it is crucial to check that the received keys are indeed generated by G. Checking that they lie in the elliptic curve is quite simple: we just have to check that their coordinate satisfy the equation defining the elliptic curve. But to show that both are generated by G, we need that n is prime, that h is coprime with n, and the following result.

Lemma 4.4. *Let E be a group with neutral element $\mathcal{O}$ and $G \in E$. We assume that $\#E = nh$, that G has order n, that n is prime, and that h is coprime with n. For all $Q \in E$, we have that Q is generated by G if and only if $nQ = \mathcal{O}$.*

Proof. Clearly, if Q is generated by G, we have $Q = xG$ for some integer x so $nQ = xnG = x\mathcal{O} = \mathcal{O}$.

We now want to prove that is $nQ = \mathcal{O}$, then Q is generated by G. We assume $nQ = \mathcal{O}$. For this, let f be the function from the group $\mathbf{Z}_n^2$ to E defined by $f(a, b) = aG + bQ$. Clearly, f is a group homomorphism. The image set of f is a subgroup of G so its order must divide $\#E$. If f is injective, the image set of f is isomorphic to $\mathbf{Z}_n^2$ so has order n^2. We know that n^2 does not divide $\#E = hn$, so f is not injective. Hence, there exists $(a, b) \in \mathbf{Z}_n^2$ such that $(a, b) \neq (0, 0)$ and $aG + bQ = \mathcal{O}$. We cannot have $b = 0$ because G has order n. As n is prime, b is invertible modulo n, and we have $Q = (ab^{-1} \bmod n).G$. So, Q is generated by G. □

Another example of a cryptographic scheme based on elliptic curves is the ECIES cryptosystem. (See Figure 4.2.) It is a hybrid cryptosystem, consisting of using the Diffie-Hellman protocol to derive two keys k_E and k_M. Then, the key k_E is used to encrypt the message using a symmetric encryption scheme Enc, and the key k_M is used to authenticate the message using a message authentication code MAC. Both notions will be covered in another chapter. More precisely, the secret key is an integer $k \in \mathbf{Z}_n^*$. The public key is a point $K = k.G$. To encrypt m, the sender picks $r \in \mathbf{Z}_n^*$, computes $R = r.G$ and $k_E \| k_M = \mathsf{KDF}(r.K \| \mathsf{extra}_1)$, where extra_1 includes some extra public information about the message. (This is defined in the standard.) Then, $c = \mathsf{Enc}_{k_E}(m)$ and $d = \mathsf{MAC}_{k_M}(c \| \mathsf{extra}_2)$, where extra_2 includes some other extra public information about the message requiring to be authenticated. The ciphertext is the triplet (R, c, d). To decrypt it, the receiver computes $S = k.R$ and $k_E \| k_M = \mathsf{KDF}(S \| \mathsf{extra}_1)$. Then, he checks that $d = \mathsf{MAC}_{k_M}(c \| \mathsf{extra}_2)$ and computes $m = \mathsf{Dec}_{k_E}(c)$.

The ECDSA algorithm is a digital signature algorithm based on elliptic curves. It will be presented in another chapter.

4.8 Pairing-Based Cryptography

For some pairs of elliptic curves G_1 and G_2 we can construct a function

$$e : G_1 \times G_2 \to G_T$$

to a group G_T (with multiplicative notations) with the following properties:

- e is easy to compute;

[2] See Section A.7.

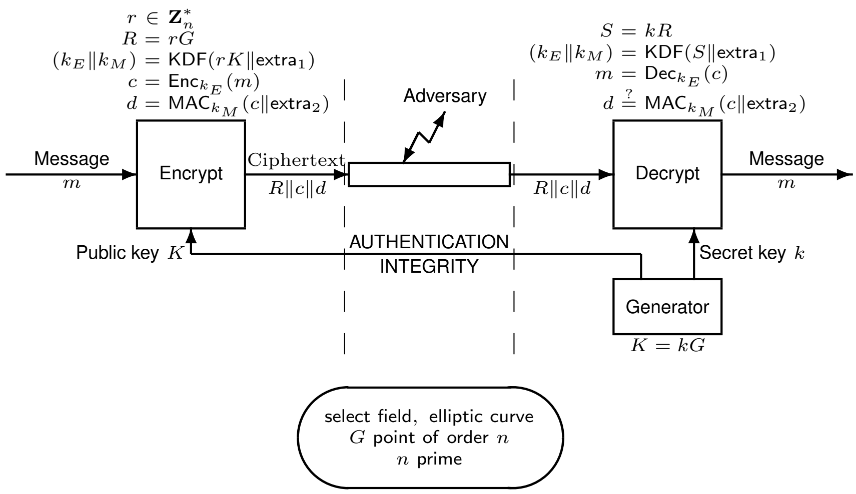

Figure 4.2: The ECIES Cryptosystem

- e is **bilinear**: $e(aP, bQ) = e(P,Q)^{ab}$ for all $a, b \in \mathbf{Z}$, $P \in G_1$, and $Q \in G_2$;
- e is **non-degenerate**: $e(P,Q) \neq 1$ for some $P \in G_1$ and $Q \in G_2$.

This construction is made possible when using *supersingular* elliptic curves which have been seen before. However, the choice of this curve is very delicate.

There are several types of pairing.

- Type-1 pairing: we have $G_1 = G_2$. This is quite common on supersingular elliptic curves. This is a common case for cryptography.
- Type-2 pairing. We have $G_1 \neq G_2$ and there exists an efficiently computable (non-degenerate) homomorphism from G_2 to G_1.
- Type-3 pairing. We have $G_1 \neq G_2$ and there exists no efficiently computable (non-degenerate) homomorphism between G_1 and G_2. This is also a common case for cryptography.
- Type-4 pairing. We have same as Type-2 with efficient hashing into $\mathcal{G}_2$. Those pairings are usually not efficient for cryptography.

With Type-1 pairing, we have $G_1 = G_2$. This can have bad consequences on the security of some cryptographic algorithms based on this curve because we can now easily distinguish if a tuple (P, A, B, C) is of form (P, xP, yP, xyP) or of form (P, xP, yP, zP) with $x, y, z \in \mathbf{Z}$. (We do so by checking that $e(A, B) = e(P, C)$.) This is the *decisional Diffie-Hellman problem* (DDH) that we have already met in a previous chapter when we solved it in the group $\mathbf{Z}_p^*$ with the help of the Legendre symbol. In such a curve $G_1 = G_2$, the decisional Diffie-Hellman problem is easy to solve but we can hope that the *computational Diffie-Hellman problem* remains hard. This is the *gap Diffie-Hellman problem*. If it is so, the group is called a *gap group*. Cryptographic constructions on those curves should mind this.

Otherwise, the availability of this function e gave birth to new cryptographic constructions:

- Joux 2000 [47]: a 3-party Diffie-Hellman key agreement using only one communication round (there existed other protocols using more rounds without pairing);
- Boneh-Franklin 2001 [24]: identity-based encryption (this was the first efficient construction);
- Boneh-Lynn-Shacham 2003 [25]: a signature scheme with short signatures (the signature is only one point in an elliptic curve);

- Bonneh-Boyen 2004 [22, 23]: a signature scheme which is secure without using hashing (namely, without a random oracle);
- Sahai-Water 2004 [72]: attribute-based encryption (in which users have attributes, and allowing to broadcast a ciphertext to a set of users sharing the same attribute).

We only detail the first construction in this chapter. The Boneh-Franklin IBE will be given in Chapter 8. The Boneh-Lynn-Shacham and Boneh-Boyen signatures will be given in Chapter 7.

Let G generate a subgroup of prime order p of $G_1 = G_2$ such that $e(G, G) \neq 1$.

- Alice picks $a \in \mathbf{Z}_p$ and broadcasts $A = aG$;
- Bob picks $b \in \mathbf{Z}_p$ and broadcasts $B = bG$;
- Charlie picks $c \in \mathbf{Z}_p$ and broadcasts $C = cG$;
- all compute $K = e(G, G)^{abc}$:
  - Alice computes $e(B, C)^a = K$;
  - Bob computes $e(C, A)^b = K$;
  - Charlie computes $e(A, B)^c = K$.

Chapter 5

Symmetric Encryption

This chapter presents block ciphers, stream ciphers, and modes of operation. It also introduce elementary algorithms to use brute force for key recovery attacks.

5.1 A Cryptographic Primitive

In this chapter, we consider encryption algorithms which use the same key to encrypt and to decrypt. We have already seen the Vernam cipher, requiring a fresh key for every new encryption. Of course, there are other types of encryption in which we can reuse a key. We will distinguish two categories: block ciphers and stream ciphers.

5.2 Block Ciphers

Block ciphers encrypt/decrypt data by *blocks* of fixed length. Typically, 64 bits or 128 bits.

DES. The DES block cipher (now, supposed to be obsolete but still widely used) was published as a US standard in 1977 [6]. It encrypts blocks of 64 bits with a key of 56 effective bits (actually, the key has 64 bits but one bit per byte is used for the checksum). Internally, the 56-bit key is expanded into a number of 16 48-bit subkeys. The encryption goes through 16 rounds, each of which uses one subkey as a round key. The rounds follow the *Feistel scheme*: the block is split into two halves. The right half goes through a round function with the round key. The output of this round function is XORed to the left half. Then, the two halves are exchanged before the next round starts. In the last round, the exchange of halves is omitted. As an illustration, Fig. 5.1 depicts a Feistel scheme with 3 rounds.

In the case of DES, the first *round function* F takes 32 bits of a half-block and the 48-bit round key K_1 to produce 32 bits which are XORed onto the other half-block.

There are some nice things with the Feistel scheme. First of all, whatever round function F, this defines a transform which is invertible (which is what we want since we want to be able to decrypt). This is shown by just constructing the inverse function. Second, the inverse transform is actually another Feistel scheme with the same round function in which we just reverse the order of the round keys. For instance, consider the 3-round Feistel scheme on Fig. 5.1 with the round functions F_{K_1}, F_{K_2}, then F_{K_3} in this order. By applying it on any input block, then applying to the result of this computation another Feistel scheme with the round functions F_{K_3}, F_{K_2}, then F_{K_1}, we obtain the original input block. This can be shown by seeing that the output of the two F_{K_3} will XOR the same value at the same place and cancel each other. Then, the two halves exchange and cancel each other. Then, the two F_{K_2} applications will XOR the same value at the same place, etc. So, the hardware to implement the decryption algorithm is just the same as the hardware to implement the encryption algorithm in which we just have to feed the round keys in a reverse order.

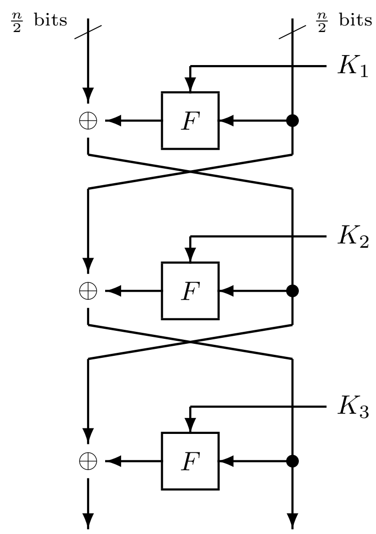

Figure 5.1: A 3-Round Feistel Scheme

Since 56 bits for a secret key are considered as too short, people considered triple encryption. This defined the *triple-DES* standard. There are two variants: triple-DES with two keys K_1 and K_2 (so, 112 bits in total), and triple-DES with three keys K_1, K_2, and K_3 (so, 168 bits in total). The latter is defined by

$$\mathsf{3DES}_{K_1,K_2,K_3}(X) = \mathsf{DES}_{K_3}\left(\mathsf{DES}^{-1}_{K_2}\left(\mathsf{DES}_{K_1}\left(X\right)\right)\right)$$

With two keys, we just use $K_1 = K_3$. For the regular DES, we just use $K_1 = K_2 = K_3$ to have backward-compatibility.

A block cipher should be secure against *key recovery* and *decryption attacks*. For both attack goals, we may consider different scenarios: *ciphertext only attacks* (i.e., the adversary only gets ciphertexts), *known plaintext attacks* (i.e., the adversary gets some (x, y) pairs where x is random and y is the encryption of x), *chosen plaintext attacks* (i.e., the adversary has access to an encryption black box to which he could submit any x and get the encryption of it in return), and *chosen ciphertext attacks* (i.e., the adversary has access to a decryption black box in addition to an encryption black box). In the latter case, for decryption attacks, the adversary is not allowed to use the decryption black box to decrypt the ciphertext he is supposed to decrypt. He can use it for any other ciphertext though.

There are many attacks known against DES. A few weak keys were identified (these do a poor job in encrypting messages). The exhaustive search with optimized hardware was proposed by Hellman in 1980. It was implemented in 1998. (A key recovery took 4 days.) Biham and Shamir invented differential cryptanalysis and did a key recovery using 2^{47} chosen plaintexts. Matsui developed linear cryptanalysis to make key recovery using 2^{43} known plaintexts. This was later optimized by Junod in 2001, using a bit less than 2^{40} known plaintexts. There are some attacks on 3DES as well. These are not covered in this lecture.

AES. The new standard is called *Advanced Encryption Standard (AES)*. It was published in 2001 [5]. It encrypts blocks of 128 bits using keys of 128, 192, or 256 bits. Its structure consists of a keylength-dependent number of rounds (Nr = 10, 12, or 14 rounds, respectively), in which a round key is used. So, the secret key is transformed into a sequence of round keys $W_0, \ldots, W_{\mathsf{Nr}}$.

In AES, a message block and a round key are represented as a 4×4 matrix of bytes (i.e., 128 bits in total). Each byte actually represents an element of $\mathsf{GF}(2^8)$ with reference polynomial $P(X) = X^8 + X^4 + X^3 + X + 1$. I.e., a bitstring $a_7 \cdots a_1 a_0$ represents the polynomial $a = a_7 X^7 + \cdots + a_1 X + a_0$ and additions and multiplications are done modulo 2 and modulo $P(X)$.

So, the addition in the field corresponds to the XOR of the bitstrings. Multiplication by the byte `0x01` is trivial: $a \times \texttt{0x01} = a$. We have already seen how to multiply a by the byte `0x02`: we just shift the byte a by one bit the the left and XOR to `0x1b` if there is a carry bit. Finally, to multiply by `0x03`, we can just multiply by `0x01` and by `0x02` and add (i.e., XOR) the two results. In AES, we only need to multiply by `0x01`, `0x02`, and `0x03`.

Each round consists of four types of successive transform: AddRoundKey which adds (i.e., XOR) the round key to the block; SubBytes which substitutes every byte a by the byte $S(a)$, following a table S (called the S-box); ShiftRows which consists of a circular shift of every row of the block by a variable number of positions; and MixColumns which consists of multiplying all columns of the block to the left by the matrix

$$M = \begin{pmatrix} \texttt{0x02} & \texttt{0x03} & \texttt{0x01} & \texttt{0x01} \\ \texttt{0x01} & \texttt{0x02} & \texttt{0x03} & \texttt{0x01} \\ \texttt{0x01} & \texttt{0x01} & \texttt{0x02} & \texttt{0x03} \\ \texttt{0x03} & \texttt{0x01} & \texttt{0x01} & \texttt{0x02} \end{pmatrix}.$$

So, we only need the multiplication by `0x01`, `0x02`, and `0x03`. More precisely, the AES encryption of a block s with a sequence of subkeys $W_0, \ldots, W_{\mathsf{Nr}}$ is implemented as follows.

AES encryption(s, W)

AddRoundKey(s, W_0)
for $r = 1$ to $\mathsf{Nr} - 1$ **do**
    SubBytes(s)
    ShiftRows(s)
    MixColumns(s)
    AddRoundKey(s, W_r)
end for
SubBytes(s)
ShiftRows(s)
AddRoundKey(s, W_{Nr})

with the following subroutines:

SubBytes(s)

for $i = 0$ to 3 **do**
    for $j = 0$ to 3 **do**
        $s_{i,j} \leftarrow S(s_{i,j})$
    end for
end for

ShiftRows(s)

replace $[s_{1,0}, s_{1,1}, s_{1,2}, s_{1,3}]$ by $[s_{1,1}, s_{1,2}, s_{1,3}, s_{1,0}]$
replace $[s_{2,0}, s_{2,1}, s_{2,2}, s_{2,3}]$ by $[s_{2,2}, s_{2,3}, s_{2,0}, s_{2,1}]$
replace $[s_{3,0}, s_{3,1}, s_{3,2}, s_{3,3}]$ by $[s_{3,3}, s_{3,0}, s_{3,1}, s_{3,2}]$

AddRoundKey(s, k)

for $i = 0$ to 3 **do**
    for $j = 0$ to 3 **do**
        $s_{i,j} \leftarrow s_{i,j} \oplus k_{i,j}$
    end for
end for

MixColumns(s)

for $i = 0$ to 3 **do**
    let v be the 4-dimensional vector with coordinates $s_{0,i}s_{1,i}s_{2,i}s_{3,i}$
    replace $s_{0,i}s_{1,i}s_{2,i}s_{3,i}$ by $M \times v$
end for

Interestingly, the function S in SubBytes is an affine transformation of the function $x \mapsto x^{254}$ in $\mathsf{GF}(2^8)$. (Note that for $x \neq 0$, $x^{254} = x^{-1}$.) But we rather keep the full table of S instead of applying the formula.

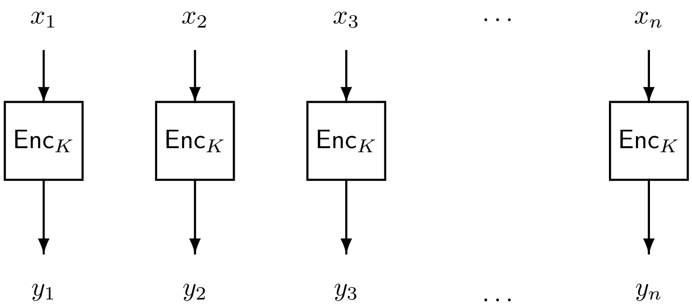

Figure 5.2: The Electronic Codebook (ECB) Mode of Encryption

To decrypt, we just have to invert all subroutine processes. This is quite straightforward, except for $\mathsf{MixColumns}$ for which the matrix has to be replaced by its inverse

$$M^{-1} = \begin{pmatrix} \texttt{0x0e} & \texttt{0x0b} & \texttt{0x0d} & \texttt{0x09} \\ \texttt{0x09} & \texttt{0x0e} & \texttt{0x0b} & \texttt{0x0d} \\ \texttt{0x0d} & \texttt{0x09} & \texttt{0x0e} & \texttt{0x0b} \\ \texttt{0x0b} & \texttt{0x0d} & \texttt{0x09} & \texttt{0x0e} \end{pmatrix}.$$

So, we now have to learn how to multiply by `0x09`, `0x0b`, `0x0d`, and `0x0e`. Here is a decryption algorithm:

AES decryption(s, W)

$\mathsf{AddRoundKey}(s, W_{\mathsf{Nr}})$
for $r = \mathsf{Nr} - 1$ down to 1 **do**
    $\mathsf{InvSubBytes}(s)$
    $\mathsf{InvShiftRows}(s)$
    $\mathsf{AddRoundKey}(s, W_r)$
    $\mathsf{InvMixColumns}(s)$
end for
$\mathsf{InvSubBytes}(s)$
$\mathsf{InvShiftRows}(s)$
$\mathsf{AddRoundKey}(s, W_0)$

Modes of operation. Once we have a block cipher, we can encrypt/decrypt a message block. If we want to encrypt a message which consists of several blocks (or even fractions of blocks), we need to plug the block cipher into a *mode of operation*. The principle of these modes of operation is that we can encrypt/decrypt messages "on-the-fly", without buffering some important part of it, but still be able to treat variable-length messages. Some modes of operation require an *initial vector* IV, as we will see. All modes internally split the plaintext into a sequence of blocks before processing.

The most straightforward mode of operation is the *Electronic Codebook (ECB) mode*. It consists of encrypting each block separately, using the block cipher. (See, Fig. 5.2.) This is however insecure for most of applications: indeed, in the messages that applications want to encrypt, it is very likely that some blocks of data repeat. For instance, in an image file representing a picture on a uniform background, the blocks which are only filled with background pixels will repeat, and their position will draw a uniform background on the ciphertext picture. The non-background blocks will look random and make the picture appear like a phantom. Since each block goes through the same encryption process, the equality of these blocks can be seen by looking at the sequence of ciphertext blocks. This ECB mode should only be used in very rare cases.

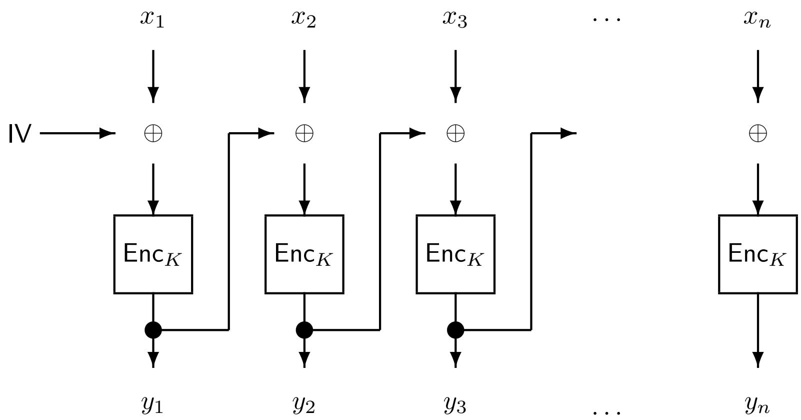

Figure 5.3: The Cipher Block Chaining (CBC) Mode of Encryption

In the *Cipher Block Chaining (CBC) mode*, each block of plaintext is XORed to the previous ciphertext block before being encrypted. The first plaintext block is XORed to an initial vector IV. So, if the message is the sequence of blocks $x_1, \ldots, x_n$, the ciphertext is the sequence $y_1, \ldots, y_n$ (sometimes with IV, as detailed below), where $y_i = \mathsf{Enc}_K(x_i \oplus y_{i-1})$, $i = 2, \ldots, n$, and $y_1 = \mathsf{Enc}_K(x_1 \oplus \mathsf{IV})$, where Enc_K is the block cipher. (See, Fig. 5.3.) There are three ways to use IV:

- use a constant, publicly known IV (e.g., $\mathsf{IV} = 0$);
- use a secret IV (so, the secret key becomes (IV, K));
- use a fresh random IV for every message x and add it as a part of the ciphertext (so, the ciphertext becomes $(\mathsf{IV}, y_1, \ldots, y_n)$).

The first two methods may suffer from similar problems as the ECB mode when it comes to look at y_1 through many encryptions. The first one is definitely not a good idea, although it is being used in applications (such as the biometric passport). In the second method, we may use a stateful encryption algorithm in which the IV for the next message becomes y_n. (So, the secret IV is used only once and we treat the sequence of messages to be encrypted as a unique stream of data to be encrypted.) It is safe if the key is set only once and the state IV keeps updating, like in the TLS standard (before version 1.3). But this option may suffer from insecurity problems as a chosen plaintext attack could select the message to encrypt based on the predictable IV.

The CBC mode also has the interesting property to be inherently immune to errors. For instance, if y_i is corrupted during transmission, decryption will mess up x_i and x_{i+1} but all subsequent blocks will be correct! Similarly, if y_i is lost, the decryption will result in one block missing and one block incoherent, at the place of the missing block.

The *Output Feedback (OFB) mode* uses an IV. It consists of defining the sequence $k_i = \mathsf{Enc}_K(k_{i-1})$, $i = 2, \ldots$, and $k_1 = \mathsf{Enc}_K(\mathsf{IV})$, and to treat the stream $k_1, \ldots$ as a one-time-pad key to encrypt x. (See, Fig. 5.4.) Clearly, this requires IV to be unique, due to the properties of one-time-pad. So, we call IV a *nonce*.[1] We can either use a random one (but by making sure that the probability to repeat an IV is negligible) or a counter-based one. In the case of a random IV, it can either be privately synchronized between the sender and the receiver, or sent in clear (so, part of the ciphertext). In the case of a counter-based IV, we can either assume that the sender and the receiver are synchronized on the counter, or just send the IV in clear again. The OFB mode works even if the last plaintext block is incomplete. This is an advantage over other encryption modes.

[1] This name is the contraction of *number* and *once*: a number which can be used only once.

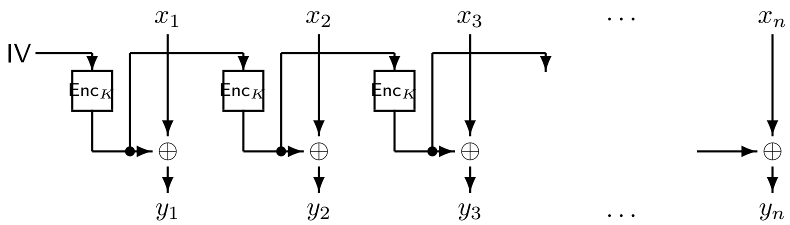

Figure 5.4: The Output Feedback (OFB) Mode of Encryption

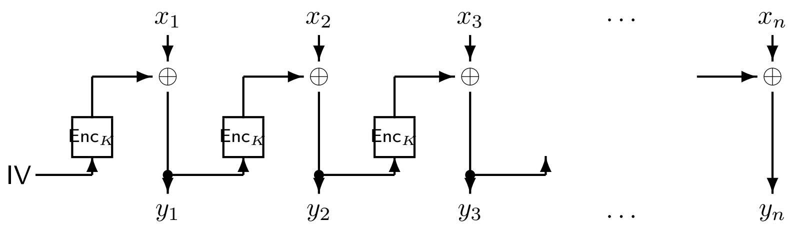

Figure 5.5: The Cipher Feedback (CFB) Mode of Encryption

The *Cipher Feedback (CFB) mode* is defined by $y_i = x_i \oplus \mathsf{Enc}_K(y_{i-1})$, $i = 2, \ldots, n$, and $y_1 = x_1 \oplus \mathsf{Enc}_K(\mathsf{IV})$. (See, Fig. 5.5.) The nonce IV options are the same as for the OFB mode. The CFB mode works even if the last plaintext block is incomplete.

The *Counter (CTR) mode* uses a nonce t_i for every block. The encryption of x_i is $y_i = x_i \oplus \mathsf{Enc}_K(t_i)$. (See, Fig. 5.6.) As its name indicates, the nonce t_i is based on a counter: it encodes a counter for the block to encrypt and a counter for the message to encrypt. Again, the CTR mode works even if the last plaintext block is incomplete.

There exists many other modes of operation. A quite popular one, which is used to encrypt hard disks, is the *XEX-based tweaked-codebook mode (TCB) with ciphertext stealing (CTS)*, which is shortened to XTS [7].[2] It assumes that the hard disk is split into sectors and that sectors are split into several blocks and up to one incomplete block, the incomplete one being the last one. It uses two keys K_1 and K_2. Except for the last two blocks of a sector, the jth block $x_{i,j}$ of the ith sector is encrypted into

$$y_{i,j} = \mathsf{Enc}_{K_1}(x_{i,j} \oplus t_{i,j}) \oplus t_{i,j}$$

where $t_{i,j} = \alpha^j \times \mathsf{Enc}_{K_2}(i)$ (see Fig. 5.7), α being a constant in the XTS standard, and the operations being done in a GF structure. So, this encryption looks like the CTR mode, so far. For the last two blocks x_{i,n_i-1} and x_{i,n_i} of a sector, it works the same if x_{i,n_i} is a complete block.

[2]It was meant to be called XTC for XEX-TCB-CTS, but this acronym was already used to refer to ecstasy drug.

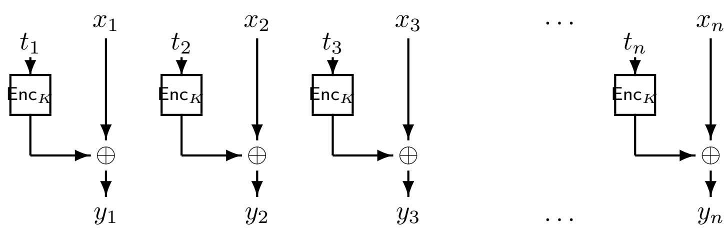

Figure 5.6: The Counter (CTR) Mode of Encryption

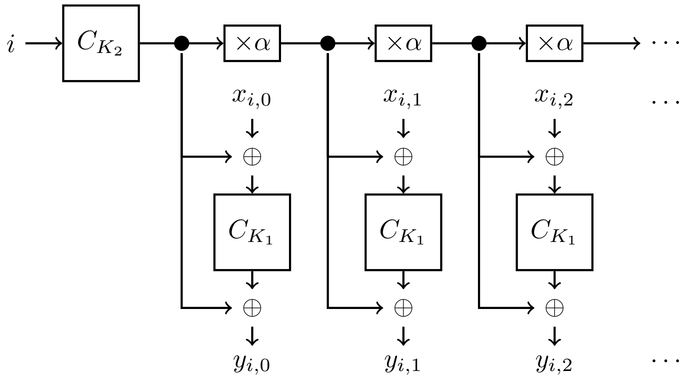

Figure 5.7: The XTS Mode of Encryption

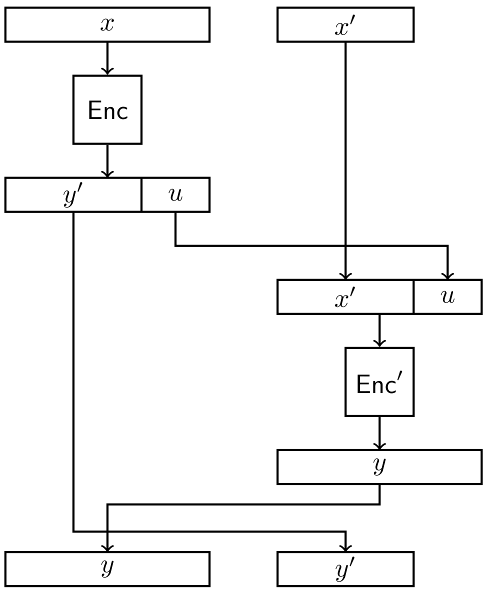

Figure 5.8: Ciphertext Stealing

Otherwise, we employ the *ciphertext stealing* trick: we compute $\mathsf{Enc}_{K_1}(x_{i,n_i-1} \oplus t_{i,n_i-1}) \oplus t_{i,n_i-1} = y_{i,n_i} \| u$ to define an incomplete block y_{i,n_i} of same length as x_{i,n_i}, and $y_{i,n_i-1} = \mathsf{Enc}_{K_1}((x_{i,n_i} \| u) \oplus t_{i,n_i}) \oplus t_{i,n_i}$ (see Fig. 5.8).

5.3 Stream Ciphers

Stream ciphers are used to encrypt streams of data on the fly. Typically, we encrypt bit-by-bit, or byte-by-byte. The main principle is that we use one-time-pad with a pseudorandom key-stream, defined from a secret key and a nonce. Again, either the nonce is based on a counter, and we may assume synchronization, or send the nonce in clear with the ciphertext, or the nonce is random, and sent in clear with the ciphertext, or the nonce is secret, but used only once.

RC4. One of the most popular stream ciphers is RC4, designed by Ronald Rivest in 1994. It was originally a trade secret, but it came in so many implementations that it is now well-known.

It generates a key-stream of bytes from a secret key (to be used only once) which is a sequence of bytes of total length between 40 bits and 256 bits. It is based on operations in the $\mathbf{Z}_{256}$ group. Let ℓ be the length (in bytes) of the key $k_0, \ldots, k_{\ell-1}$. RC4 starts with a KSA algorithm defined by

$j \leftarrow 0$
for $i = 0$ to 255 **do**
    $S[i] \leftarrow i$
end for
for $i = 0$ to 255 **do**
    $j \leftarrow j + S[i] + k_{i \bmod \ell}$
    swap $S[i]$ and $S[j]$
end for

Except the $i \bmod \ell$ operation, all others are to be considered modulo 256. Then, the PRGA (pseudorandom generator algorithm) generates the stream:

$i \leftarrow 0$
$j \leftarrow 0$
loop
    $i \leftarrow i + 1$
    $j \leftarrow j + S[i]$
    swap $S[i]$ and $S[j]$
    output $S[S[i] + S[j]]$
end loop

As we can see, the RC4 algorithm is an automaton based on a *state*, defined by two bytes i and j, and a byte permutation S.

In the TLS protocol (which is discussed in Section A.5), the key is used only once, but the final state after the encryption of a message is kept to serve as the initial (PRGA) state of the encryption of the next message. So, we need synchronization as encryption is stateful. In the WEP protocol (which is discussed in Section A.1, the first three bytes of the key define a nonce to be sent in clear, and the rest is the actual secret key.

There are many known weaknesses in RC4. For instance, some correlations between some key bytes and some output bytes allow to make a passive key recovery attack in WEP. The bad distribution of output bytes allows to make a ciphertext-only decryption attack in TLS when the same plaintext is encrypted many times (this is the case with *secure http cookies*). There are speculations about state agencies being able to fully break RC4. As a result, WEP is obsolete and RC4 is now prohibited in TLS.

A5/1. Another very popular stream cipher, as it is used in GSM communication (which is presented in Section A.3, is the A5/1 algorithm. Again, it was a trade secret but was reverse engineered (and subsequently broken). It uses a 64-bit secret and a 22-bit counter, used as a nonce. The key and the counter are first transformed into an initial state. Then, an automaton based on asynchronous linear feedback shift registers generates a key-stream of bits.

There are many attacks against A5/1. For instance, there are key-recovery known plaintext attacks. There are also active attacks on GSM (attacks which force mobile equipments to change the cipher by keeping the same secret key). There are also ciphertext-only recovery attacks using (optimized) bruteforce. Consequently, we should not expect too much about the privacy of our GSM communication!

5.4 Bruteforce Inversion Algorithms

Let $\mathcal{K}$ be a set of given size N. Consider the *random key guessing game* during which a challenge selects a key $K \in \mathcal{K}$ at random, then an adversary makes guesses for the value of K until it is correct. The game is formalized as follows:

Game:
pick $K \in_D \mathcal{K}$
$\mathcal{A}^{\mathcal{O}} \to k$
return $1_{K=k}$

$\mathcal{O}(\mathsf{query})$:
return $1_{K=\mathsf{query}}$

The adversary knows the set $\mathcal{K}$ and may know the sampling distribution of K. This distribution may be uniform or arbitrary. For instance, someone trying to open a safe without knowing the combination to open it would just make some guesses until the safe opens.

In the uniform distribution case, the adversary can just enumerate the elements of $\mathcal{K}$ until the value K is found. In terms of trials, the worst case complexity is N. The average case complexity is

$$\sum_{i=1}^{N} \frac{i}{N} = \frac{N+1}{2}$$

When the distribution is arbitrary and unknown, the best strategy is to enumerate the values of $\mathcal{K}$ in a random order to reach the same average complexity. If the distribution is known, we can enumerate the values of $\mathcal{K}$ by decreasing order of likelihood and obtain the optimal average complexity which is called the *guesswork entropy*.

The game may be different if the adversary is given a clue which we call a *witness* w. Then, the optimal strategy is to enumerate all $k \in \mathcal{K}$ by decreasing order of $\Pr[K = k|w]$. But then, our strategies must consider the complexity in terms of trials, computation time, memory space, and in terms of the probability of success as well.

In offline key recovery, the adversary can make only one guess. He does so based on a witness $w = F(K)$. The function F may be deterministic or not. For instance, if the adversary is running a chosen plaintext attack, $F(K)$ consists of the ciphertexts which are obtained by the (fixed) chosen plaintexts. With a deterministic function F, we can consider the offline key recovery game (on the left) and a variant which consists of finding *any* key matching the clue (on the right):

Game:
pick $K \in_D \mathcal{K}$
$W \leftarrow F(K)$
$\mathcal{A}(W) \to k$
return $1_{k=K}$

Game:
pick $K \in_D \mathcal{K}$
$W \leftarrow F(K)$
$\mathcal{A}(W) \to k$
return $1_{F(k)=W}$

Let us consider the case of a password-based access control scheme in which the password K has a witness $w = F(K)$ which is stored in clear. In some concrete protocols, we may have $F(K) = \mathsf{Enc}_K(0)$, for instance. The access control consists in hashing (by F) the typed password and to compare the result with w. Access is granted if it matches. In this case, the goal of the adversary is to find, from w, *any* password k such that $w = F(k)$ (but not necessarily the correct one). The adversary could just enumerate all k's until a solution is found. If M is the size of the output range of F and F is a random function, every new trial k is a solution with probability $\frac{1}{M}$. For $M \ll N$, the expected complexity is

$$\sum_{i\geq 0} i \Pr[i \text{ } \mathsf{trials}] = \sum_{i\geq 0} \Pr[> i \text{ } \mathsf{trials}] = \sum_{i\geq 0} \left(1 - \frac{1}{M}\right)^i = M$$

For $M \gg N$, collisions on F are rare and the complexity is similar than in the previous case, so close to $\frac{N}{2}$.

A *dictionary attack* consists of preparing a complete table for the inverse function $w \mapsto F^{-1}(w)$. That is, we prepare a sorted list of all $(F(k), k)$. The attack then works with constant complexity, but requires a memory of $\mathcal{O}(N)$, and a preprocessing time of $\mathcal{O}(N)$ as well. The probability of success is 1.

With an incomplete dictionary of size D, the precomputation time is $\mathcal{O}(D)$, the memory complexity is $\mathcal{O}(D)$, the time complexity of the attack phase is $\mathcal{O}(1)$, but the probability of success is $\frac{D}{N}$. As we can see, already, we may consider tradeoffs.

The attack model can be enriched by considering a multi-target version: instead of targeting a single K, the goal maybe to recover at least one K_i from a list $K_1, \ldots, K_T$ of T targets. In that case, the dictionary attack needs a precomputation time of $\mathcal{O}(D)$, a memory complexity of $\mathcal{O}(D)$, a time complexity of the attack phase of $\mathcal{O}(T)$, but the probability of success is $1 - \left(1 - \frac{D}{N}\right)^T \approx 1 - e^{-\frac{DT}{N}}$. This is quite interesting for $D \approx T \approx \sqrt{N}$: we have a succeeding attack of overall complexity $\mathcal{O}(\sqrt{N})$ (instead of the regular $\mathcal{O}(N)$) on $\Omega(\sqrt{N})$ targets. Here is a formalization of the multitarget inversion game with a partial dictionary attack.

Game:
setup F
$\mathcal{A}_1^F \to \mathsf{dict}$
pick $K_1, \ldots, K_T \in_D \mathcal{K}$
$w_i \leftarrow F(K_i)$ for $i = 1, \ldots, T$
$\mathcal{A}_2^F(\mathsf{dict}, w_1, \ldots, w_T) \to (i, k)$
return $1_{F(k)=w_i}$

$\mathcal{A}_1^F$: (preprocessing)
for D candidates k **do**
    compute $F(k)$
    $\mathsf{dict}\{F(k)\} \leftarrow k$
end for
return dict

$\mathcal{A}_2^F(\mathsf{dict}, w_1, \ldots, w_T)$: (attack)
for $i = 1$ to T **do**
    if $\mathsf{dict}\{F(w_i)\}$ exists **then**
        return $(i, \mathsf{dict}\{F(w_i)\})$
    end if
end forreturn $\perp$

When the witness function is not deterministic, the dictionary attack is more complicated, if not impossible. For instance, in password access control, if the password is hashed together with a random *salt* which is stored with the hash, the dictionary must be dedicated to this salt value (which excludes multi-target variants), or to consider all possible salts (which makes the dictionary bigger). In known plaintext attacks, the plaintexts cannot be prepared when making the dictionary. We can conclude with an offline attack to recover K such that $F(K; W_2) = W_1$ corresponding to the witness $W = (W_1, W_2)$ and the salt W_2:

select a random ordering $k_{\sigma(1)}, \ldots, k_{\sigma(N)}$ of $\mathcal{K}$
for $i = 1$ to N **do**
    if $F(k_{\sigma(i)}; W_2) = W_1$ **then**
        stop and yield $k_{\sigma(i)}$
    end if
end for
search failed

We obtain the expected complexity $\mathcal{O}(N)$ again.

5.5 Subtle Bruteforce Inversion Algorithms

Meet-in-the-middle attack on double encryption. Consider a double encryption scheme

$$\mathsf{Enc}_{K_1,K_2}(x) = \mathsf{Enc}_{K_2}\left(\mathsf{Enc}_{K_1}(x)\right)$$

where the keys belong to a set $\mathcal{K}$ of size N. We consider a known-plaintext attack scenario where a pair (x, z) with $z = \mathsf{Enc}_{K_1,K_2}(x)$ is known. We assume that this equation uniquely characterizes (K_1, K_2). (Otherwise, we just consider enough pairs instead of just one.)

The *meet-in-the-middle* algorithm (see below) consists of first preparing a dictionary of $(\mathsf{Enc}_{k_1}(x), k_1)$ pairs. Then, it makes an exhaustive search on k_2 to compute $y = \mathsf{Enc}_{k_2}^{-1}(z)$, look for (y, k_1) in the dictionary and print (k_1, k_2) if there is such an entry. Interestingly, the complexity is $\mathcal{O}(N)$ both in time and space, although the double encryption would expect to have a security of $\Omega(N^2)$.

for all k_1 **do**
    compute $y = \mathsf{Enc}_{k_1}(x)$

insert k_1 in $\mathsf{dict}\{y\}$
end for
for all k_2 **do**
compute $y = \mathsf{Enc}_{k_2}^{-1}(z)$
for all k_1 in $\mathsf{dict}\{y\}$ **do**
yield (k_1, k_2) as a possible key
end for
end for

Time-memory tradeoffs inversion. Another subtle inversion algorithm is for the single encryption case with a deterministic function F. (Actually, this can invert any deterministic function F, e.g. a hash function.) Imagine an algorithm preparing ℓ tables of m entries $(k^s_{i,0}, k^s_{i,t})$, $s = 1, \ldots, \ell$, $i = 1, \ldots, m$. For each table s, a random reduction function R_s is selected so that $R_s(F(k)) \in \mathcal{K}$ for all $k \in \mathcal{K}$. Then, for each entry in the table, $k^s_{i,0}$ is randomly selected and we compute $k^s_{i,j} = R_s(F(k^s_{i,j-1}))$, $j = 1, \ldots, t$. This runs as follows.

for $s = 1$ to ℓ **do**
pick a reduction function R_s at random and define $f_s : k \mapsto R_s(F(k))$
for $i = 1$ to m **do**
pick k' at random
$k \leftarrow k'$
for $j = 1$ to t **do**
compute $k \leftarrow f_s(k)$
end for
$T_s\{k\} \leftarrow k'$
end for
end for

Clearly, this precomputation takes time $\ell m t$ and the memory complexity is ℓm.

During the attack phase, we are given $w = F(K)$ and we want to recover K. For each s we compute $R_s(w)$ then iterate $k \mapsto R_s(F(k))$ at most t times. If one of the values matches on $k^s_{i,t}$, then we can start from the corresponding $k^s_{i,0}$, iterate again until we reach $R_s(w)$. If we do, we obtain a preimage of $R_s(w)$ by $R_s \circ F$, thus a preimage of w by F with high probability. So, we recover the key. More precisely, the algorithm runs as follows.

for $s = 1$ to ℓ **do**
set i to 0
set k to $R_s(w)$
while $T_s\{k\}$ does not exist and $i < t$ **do**
increment i
$k \leftarrow f_s(k)$
end while
if $T_s\{k\}$ exists **then**
$k' \leftarrow T_s\{k\}$
while $F(k') \neq y$ and $i < t$ **do**
increment i
$k' \leftarrow f_s(k')$
end while
if $F(k') = y$ **then**
yield k' as a possible key
end if
end if
end for

The complexity is $\mathcal{O}(\ell t)$. The optimal selection is for $\ell \approx m \approx t \approx N^{\frac{1}{3}}$ for which we obtain a complexity of $\mathcal{O}(N^{\frac{2}{3}})$.

In the case of DES where $N = 2^{56}$, this attack works with about 2^{37} operations, after having made the tables within 2^{56} operations.

5.6 Pushing the Physical Limits

As mentioned before, the number of 2007-machines needed to make an exhaustive search on a 128-bit secret key within the age of the universe is incredibly high. We may expect that the Moore's law improves this, but we shall still keep in mind that there is an energy bill. So far, erasing a single bit of memory on a machine at temperature T (in Kelvin) requires an energy of $kT \ln 2$, where

$$k = 1.38 \times 10^{-23}\ JK^{-1}$$

is the constant of Boltzmann. This is the Laudauer result from 1961 [53]. Assuming that each trial in an exhaustive search requires to erase $\log_2 N$ bits of memory at least, we can estimate the energy needed to do the exhaustive search, whatever the speed. With a machine running at $3\,\mu K$ (which is extremely cold), we need $12 \times 10^9\ J$. To do it within one second requires the full power of a last-generation nuclear powerplant.

Bennett proved in 1973 [16] that in theory, we could compute without spending energy as long as each computation step is reversible (so that we have no erasure), but this requires to accumulate all intermediary results so to waste memory.

We can make an exhaustive search in a reversible manner and save memory as follows. We define three operations which will modify the internal state $\langle x, y, k, z, t\rangle$ of the computer in a reversible manner:

$$\begin{array}{lrcl}
\text{INC:} & \langle x, y, k, z, t\rangle & \mapsto & \langle x, y, k+1, z, t\rangle \\
\text{ENC:} & \langle x, y, k, z, t\rangle & \mapsto & \langle x, y, k, z \oplus \mathsf{Enc}(k, x), t\rangle \\
\text{CMP:} & \langle x, y, k, z, t\rangle & \mapsto & \langle x, y, k, z, t \oplus k \cdot 1_{y=z}\rangle
\end{array}$$

Then, the sequence ENC, CMP, ENC, INC does in fact the operation

$$\langle x, y, k, z, t\rangle \mapsto \langle x, y, k+1, z, t \oplus k \cdot 1_{y \oplus z = \mathsf{Enc}(k,x)}\rangle$$

If we execute this operation 2^{128} times on the initial state $\langle x, y, 0, 0, 0\rangle$, we obtain the XOR of all keys such that $y = \mathsf{Enc}(k, x)$. Assuming K is unique, we get the final state $\langle x, y, 0, 0, K\rangle$.

Reversible operations can typically be implemented on a quantum computer. However, there is a better algorithm to do an exhaustive search: the Grover algorithm [43]. This algorithm allows, for a function F over a domain of size N, to find a preimage when it is unique, in a complexity corresponding to $\mathcal{O}(\sqrt{N})$ evaluations of F. This algorithm motivates using keys of 256 bits in AES. Indeed, a 128-bit key would be found with a complexity of 2^{64} on a quantum computer.

5.7 Formalism

We finish this chapter by introducing some formal definitions about symmetric encryption. First of all, we define a block cipher. For simplicity, we assume that it operates on bitstrings, as this is the most common case.

Definition 5.1. *A* ***block cipher*** *is a tuple* $(\{0,1\}^k, \{0,1\}^n, \mathsf{Enc}, \mathsf{Dec})$ *with a key domain* $\{0,1\}^k$*, a block domain* $\{0,1\}^n$*, and two* efficient *deterministic algorithms* Enc *and* Dec*. It is such that*

$$\forall K \in \{0,1\}^k \quad \forall X \in \{0,1\}^n \quad \mathsf{Dec}(K, \mathsf{Enc}(K, X)) = X$$

Write $C_K(\cdot) = \mathsf{Enc}(K, .)$ *and* $C_K^{-1}(\cdot) = \mathsf{Dec}(K, .)$.

So, k is the key length and n is the message block length. We let the notion of *efficient algorithm* undefined here. The main point is that we are not interested in block ciphers which have too slow or too expensive implementations.

It is important that encryption and decryption are deterministic.

As an equivalent definition, we can just say that C_K and C_K^{-1} are defined as permutations over the block space which are inverse of each other for every key $K \in \{0,1\}^k$.

Block ciphers are extended into variable-length encryption schemes. Most commonly, these encryption schemes are *length-preserving* in the sense that the ciphertext and the plaintext have always the same length. We could define the encryption over the set of all bitstrings, but sometimes, some message length are not made available. So, we must define the encryption scheme over a subset $\mathcal{D}$ of the set of bitstrings $\{0,1\}^*$.

Definition 5.2. *A (variable-length, length-preserving) symmetric encryption scheme is defined by a tuple* $(\{0,1\}^k, \mathcal{D}, \mathsf{Enc}, \mathsf{Dec})$ *with a key domain* $\{0,1\}^k$*, a plaintext domain* $\mathcal{D} \subseteq \{0,1\}^*$*, and two* efficient *deterministic algorithms* Enc *and* Dec*.*

It is such that

$$\forall K \in \{0,1\}^k \quad \forall X \in \mathcal{D} \quad \begin{cases} \mathsf{Dec}(K, \mathsf{Enc}(K,X)) & = & X \\ |\mathsf{Enc}(K,X)| & = & |X| \end{cases}$$

Write $C_K(\cdot) = \mathsf{Enc}(K,.)$ *and* $C_K^{-1}(\cdot) = \mathsf{Dec}(K,.)$*.*

In the definition, we denote by $|X|$ the length of X. So, $\mathsf{Enc}(K,X)$ and X always have the same length.

We have seen that some modes of operation and stream ciphers have an extra input which is used as a *nonce*. So, we must extend our definition.

Definition 5.3. *A (nonce-based, variable-length, length-preserving) symmetric encryption scheme is a tuple* $(\{0,1\}^k, \mathcal{D}, \mathcal{N}, \mathsf{Enc}, \mathsf{Dec})$ *with a key domain* $\{0,1\}^k$*, a plaintext domain* $\mathcal{D} \subseteq \{0,1\}^*$*, a nonce domain* $\mathcal{N}$*, and two* efficient *deterministic algorithms* Enc *and* Dec*.*

It is such that

$$\forall K \in \{0,1\}^k \quad \forall X \in \mathcal{D} \quad \forall N \in \mathcal{N} \quad \begin{cases} \mathsf{Dec}(K,N,\mathsf{Enc}(K,N,X)) = X \\ |\mathsf{Enc}(K,N,X)| = |X| \end{cases}$$

N is supposed to be used only once for encryption. We could use a random nonce (we have to be careful about values which could repeat with a random selection by having nonces large enough), or a counter. We could send the nonce in clear with the ciphertext or have the sender and receiver synchronized on the nonce by other means. Some modes of operation use an IV which can be used as a nonce, but an IV which is secret does not fit this notion of a nonce.

Once we defined the encryption scheme with the correctness notion (namely, that the decryption of the ciphertext is equal to the plaintext), we must define the security notion. First of all, we define the security against key recovery, either by chosen plaintext attack or by chosen plaintext and ciphertext attack. (We could easily define the security by known plaintext attacks or by ciphertext only attacks.) In the following security notion, we see that there is a key K selected at random. The adversary $\mathcal{A}$ can play with the encryption function as a black box and he must guess the value of K.

Definition 5.4. *A symmetric encryption scheme* $(\{0,1\}^k, \mathcal{D}, \mathcal{N}, \mathsf{Enc}, \mathsf{Dec})$ *is* (q,t,ε)-secure against key recovery under chosen plaintext attacks (CPA) *if for any algorithm* $\mathcal{A}$ *limited to a time complexity* t *and to* q *queries, the advantage* Adv *is bounded by* ε*, where*

$$\mathsf{Adv} = \Pr[\textit{game returns } 1]$$

Game
1: $K \xleftarrow{\$} \{0,1\}^k$
2: $\mathsf{Used} \leftarrow \emptyset$
3: $\mathcal{A}^{\mathsf{OEnc}} \to K'$
4: ***return*** $1_{K=K'}$

Oracle $\mathsf{OEnc}(N,X)$*:*
5: ***if*** $N \in \mathsf{Used}$ ***then return*** $\perp$
6: $\mathsf{Used} \leftarrow \mathsf{Used} \cup \{N\}$
7: ***return*** $\mathsf{Enc}(K,N,X)$

Security against key recovery under chosen plaintext/ciphertext attacks (CPCA) *is similar with the game*

Game
$K \xleftarrow{\$} \{0,1\}^k$
$\mathsf{Used} \leftarrow \emptyset$
$\mathcal{A}^{\mathsf{OEnc},\mathsf{ODec}} \to K'$
return $1_{K=K'}$

Oracle $\mathsf{OEnc}(N,X)$:
if $N \in \mathsf{Used}$ ***then return*** $\perp$
$\mathsf{Used} \leftarrow \mathsf{Used} \cup \{N\}$
return $\mathsf{Enc}(K,N,X)$

Oracle $\mathsf{ODec}(N,Y)$:
return $\mathsf{Dec}(K,N,Y)$

We say that $\mathcal{A}$ is *nonce-respecting* if it never make two encryption queries using the same nonce. However, he could make decryption queries by using the same nonce many times.

In the definition, the notation $\mathsf{Enc}(K,.,.)$ as a superscript of $\mathcal{A}$ means that the algorithm $\mathcal{A}$ has access to an external black box such that by feeding it with a pair (N,X), it returns $\mathsf{Enc}(K,N,X)$ in one unit of time. Similarly, the superscript $\mathsf{Enc}(K,.,.),\mathsf{Dec}(K,.,.)$ means a black box access to two oracles: $(N,X) \mapsto \mathsf{Enc}(K,N,X)$ and $(N,Y) \mapsto \mathsf{Dec}(K,N,Y)$.

Here, security is defined by having a bounded advantage ε for adversaries limited to some resources (q,t): the limitation holds in the total number of queries q to any oracle and in the time complexity t of the overall attack, i.e. the number of steps of the algorithm.

Clearly, security against chosen plaintext and ciphertext attacks implies security against chosen plaintext attacks. So, security against chosen plaintext and ciphertext attacks is a stronger security notion.

It is not always enough to have security against key recovery. Indeed, we could define an encryption scheme doing nothing (i.e., by $\mathsf{Enc}(K,N,X) = X$ for all K, N, and X) and it would be very hard to make a key recovery attack. However, no message is really encrypted. Actually, the key is not used at all. So, we define security against decryption attacks, by using another scenario: a key K, nonce N, and message X are selected at random. The adversary is given $Y = \mathsf{Enc}(K,N,X)$ and can play with the encryption as a black box. With chosen decryption attacks, he is not allowed to query the decryption oracle $\mathsf{Dec}(K,.,.)$ with the (N,Y) pair. The goal is to guess the value of X.

Definition 5.5. *A symmetric encryption scheme* $(\{0,1\}^k, \mathcal{D}, \mathcal{N}, \mathsf{Enc}, \mathsf{Dec})$ *is* (q,t,ε)*-**secure against decryption** under CPA (resp. CPCA) if for any algorithm* $\mathcal{A}$ *limited to a time complexity* t *and to* q *queries, the advantage* Adv *is bounded by* ε.

$$\mathsf{Adv} = \Pr[\textit{game returns } 1]$$

Game
$K \xleftarrow{\$} \{0,1\}^k$
$X_0 \xleftarrow{\$} \mathcal{D}$, $N_0 \xleftarrow{\$} \mathcal{N}$
$\mathsf{Used} \leftarrow \{N_0\}$
$Y_0 \leftarrow \mathsf{Enc}(K,N_0,X_0)$
$\mathcal{A}^{\mathsf{OEnc}(,\mathsf{ODec})}(N_0,Y_0) \to X$
return $1_{X=X_0}$

Oracle $\mathsf{OEnc}(N,X)$:
if $N \in \mathsf{Used}$ ***then return*** $\perp$
$\mathsf{Used} \leftarrow \mathsf{Used} \cup \{N\}$
return $\mathsf{Enc}(K,N,X)$

Oracle $\mathsf{ODec}(N,Y)$:
if $(N,Y) = (N_0,Y_0)$ ***then return*** $\perp$
return $\mathsf{Dec}(K,N,Y)$

Note that $\mathcal{D}$ must be a finite set to select $X \in \mathcal{D}$ with a uniform distribution.

As an adversary guessing the key K can always decrypt Y by running himself the Dec algorithm, security against decryption attack implies security against key recovery attack. So, security against decryption attack is a stronger security notion.

Yet, we may want to protect messages against partial decryption, or consider stronger security notions. For that, we can try to compare the encryption scheme with an ideal one. An ideal encryption scheme Π would be such that for every nonce value N, $\Pi(N,.)$ would be a uniformly distributed length-preserving permutation of $\mathcal{D}$. The goal of an adversary would be to distinguish if a black-box encryption machine is using the real encryption scheme or the ideal one. We define the security against *distinguishers* as follows.

Definition 5.6. *A symmetric encryption scheme* $(\{0,1\}^k, \mathcal{D}, \mathcal{N}, \mathsf{Enc}, \mathsf{Dec})$ *is* (q,t,ε)-secure against distinguishers *under CPA (resp. CPCA) if for any algorithm* $\mathcal{A}$ *limited to a time complexity* t *and to* q *queries, the advantage* Adv *is bounded by* ε.

$$\mathsf{Adv} = \Pr[\Gamma_1 \textit{ returns } 1] - \Pr[\Gamma_0 \textit{ returns } 1]$$

Game Γ_b

1: $K \xleftarrow{\$} \{0,1\}^k$

2: *for every* N*, pick a length-preserving permutation* Π_N *over* $\mathcal{D}$

3: $\mathsf{Used} \leftarrow \emptyset$

4: $\mathcal{A}^{\mathsf{OEnc}(,\mathsf{ODec})} \rightarrow z$

5: ***return*** z

Oracle $\mathsf{OEnc}(N,X)$*:*

1: ***if*** $N \in \mathsf{Used}$ ***then return*** $\perp$

2: $\mathsf{Used} \leftarrow \mathsf{Used} \cup \{N\}$

3: ***if*** $b = 0$ ***then return*** $\Pi_N(X)$

4: ***return*** $\mathsf{Enc}(K,N,X)$

Oracle $\mathsf{ODec}(N,Y)$*:*

5: ***if*** $b = 0$ ***then return*** $\Pi_N^{-1}(Y)$

6: ***return*** $\mathsf{Dec}(K,N,Y)$

We conclude by an informal proof that indistinguishability is a stronger security notion that decryption security. We show it for chosen plaintext attacks, but it works with chosen plaintext/ciphertext attacks as well. Indeed, assuming that we have a decryption adversary $\mathcal{A}$, we can define a distinguisher $\mathcal{D}$ as follows:

$\mathcal{D}^{\mathsf{Enc}(K,.)}$:

1: pick X, query $\mathsf{Enc}(K,.)$ on X to get Y

2: run $\mathcal{A}^{\mathsf{Enc}(K,.)}(Y) \rightarrow X'$

3: output $1_{X=X'}$

The advantage of $\mathcal{D}$ is computed as

$$\begin{aligned} & \Pr[\mathcal{D}^{\mathsf{Enc}(K,.)} \rightarrow 1] - \Pr[\mathcal{D}^{\Pi(\cdot)} \rightarrow 1] \\ = \; & \Pr[\mathcal{A}^{\mathsf{Enc}(K,.)}(\mathsf{Enc}(K,X)) = X] - \Pr[\mathcal{A}^{\Pi(\cdot)}(\Pi(X)) = X] \\ \geq \; & \Pr[\mathcal{A} \text{ wins}] - \varepsilon' \end{aligned}$$

where $\Pr[\mathcal{A}^{\Pi(\cdot)}(\Pi(X)) = X] \leq \varepsilon'$. It is a bit technical that we can obtain a pretty low upper bound ε'. This is actually a combinatorial result as it does not depend at all on the Enc design. Hence, assuming security against distinguishers, we deduce that $\Pr[\mathcal{A} \text{ wins}]$ must be small, so a security against decryption attacks.

Chapter 6

Integrity and Authentication

This chapter mostly discusses on two cryptographic primitives: message authentication codes and hash functions. Some related primitives will be covered as well. *Message authentication codes* use a symmetric secret key to authenticate messages: the sender appends a tag to the message, computed based on the message and the key, and the verifier checks that the received message and tag are consistent with the key. This shall protect against an adversary trying to *forge* messages without being detected. *Hash functions* map an arbitrary bitstring to a *digest* (or *message hash*) of fixed length. There are many security properties which may be required on hash functions, depending on the application.

6.1 Commitment Scheme

In a *commitment scheme*, there are two participants, the sender and the receiver, running a protocol in two phases: the commitment phase and the opening phase. The sender wants to commit on a message X without revealing it. Typically, he picks some random r and computes $(c, k) = \mathsf{Commit}(X; r)$. He then reveals c to the receiver. In the opening phase, the sender reveals k and the receiver can compute $\mathsf{Open}(c, k) = X$. The correctness requirement implies that $\mathsf{Open}(\mathsf{Commit}(X; r)) = X$ for any X and r. As we will see, the security consists of two properties: *hiding* and *binding*.

The commitment must be *hiding*: the receiver shall not retrieve any information about X during the commitment phase (i.e., from c). This is similar to encryption.

Compared to encryption, there is a second security property which is required: the commitment must be *binding*: the sender shall not be able to construct c, k, k' such that $\mathsf{Open}(c, k) \neq \mathsf{Open}(c, k')$.

In many cases, the commitment scheme is based on a single function H: we have $\mathsf{Commit}(X; r) = (c, k)$ with $c = H(X\|r)$ and $k = (X, r)$ and $\mathsf{Open}(c, (X, r)) = X$ if $c = H(X\|r)$ and $\perp$ otherwise. As we can see, H behaves like an encryption function to *hide* X, but no decryption algorithm is needed in the scheme. In this construction, the binding property means that we cannot find X, X', r, and r' such that $H(X\|r) = H(X'\|r')$ and $X \neq X'$.

This primitive can be used to make two participants flip a coin over a communication channel in a fair way (see Fig. 6.1): Alice would flip a bit b and commit to it to Bob. Then, Bob would flip a bit b' and send it to Alice. Then, Alice would open her commitment and the outcome of the protocol would be $b \oplus b'$. If Alice is honest, since the commitment is hiding, b' must be independent from b, so $b \oplus b'$ shall be uniformly distributed. If Bob is honest, since the commitment is binding, the opened b must be independent from b', so $b \oplus b'$ shall be uniformly distributed. This can generalize to rolling a die.

There are many kinds of commitments: those requiring an interactive protocol, and those which are non-interactive (the specification in terms of Commit and Open algorithms as above is essentially non-interactive). As for the security notions, there are several degrees: perfect, statisti-

$$
\begin{array}{rcl}
\textbf{Alice} & & \textbf{Bob} \\
\text{pick } b \in \{0,1\} & & \\
\text{pick } r,\ (c,k) \leftarrow \mathsf{Commit}(X;r) & \xrightarrow{\quad c \quad} & \\
 & \xleftarrow{\quad b' \quad} & \text{pick } b' \in \{0,1\} \\
 & \xrightarrow{\quad k \quad} & b \leftarrow \mathsf{Open}(c,k) \\
z \leftarrow b \oplus b' & & z \leftarrow b \oplus b' \\
\textbf{output: } z & & \textbf{output: } z
\end{array}
$$

Figure 6.1: A Coin Flipping Protocol

cal, and computational. For instance, in a perfectly hiding commitment, X and c are statistically independent. For statistically hiding commitments, the distributions of $(c|X = x_0)$ and $(c|X = x_1)$ would be indistinguishable for any x_0 and x_1, up to some "negligible" probability. For computationally hiding commitments, this would only hold for computationally bounded adversaries. Finally, some commitments may require a global setup such as a common reference string, which is supposed to be set up once for all.

As we will see, the most practical commitment schemes are based on a hash function.

6.2 Key Derivation Function and Pseudorandom Generator

Pseudorandom generator. A *pseudorandom generator (PRNG)* is typically an automaton, initialized with a seed, which updates its state and outputs a number at every generation. Cryptographic pseudorandom generators must be such that the generated sequence of numbers must be indistinguishable from a sequence of random numbers. For instance, the stream ciphers that we have seen, with a secret key playing the role of the seed can be considered as a PRNG, since they generate a pseudorandom key stream.

There are famous failure cases related to PRNG. For instance, we can often see implementations of cryptographic schemes in which the secret keys are generated by a PRNG using a seed of too low entropy, e.g., 16 bits. In such cases, even though the secret key may be as large as 128 bits, we can do an exhaustive search on the seed to recover it. In another failure case, some random number in $\mathbf{Z}_q$ was generated by reducing an ℓ-bit random number modulo q, where $\ell = \lceil \log_2 q \rceil$. Although it looks reasonable, it introduces an enormous bias in the distribution of the generated numbers, as the ones between 0 and $2^\ell - q - 1$ would appear twice more often than others. This led to an attack against the signature algorithm DSA [4]. To defeat it, a better way consisted in reducing modulo q a 2ℓ-bit random number. This way, there are $2^{2\ell} \bmod q$ numbers appearing with a slightly larger probability, but the gap is of $2^{-2\ell}$ instead of $2^{-\ell}$.

Pseudorandom function. Quite often, we may need a function f_K set up with a key K. For instance, a block cipher is a permutation over a block space which is set up by a key. Sometimes, we just need f_K to produce values looking like random. We say that f is a *pseudorandom function (PRF)* if by playing with a black-box function, we cannot say whether the function in the black box is just f_K set up with a random K or a truly random function.

Key derivation function. A *key derivation function (KDF)* typically maps some random value with imperfect distribution into a symmetric key which has a distribution close to uniform.

6.3 Cryptographic Hash Function

A hash function maps a bitstring of arbitrary length to a bitstring of fixed length (e.g., 160 bits for the SHA1 hash function). There are three main uses of hash functions: domain expansion, commitment, and pseudorandom generation.

Domain expanders are used in digital signatures: we design signature algorithms to be able to sign bitstrings of fixed length but we also want to sign arbitrary bitstrings. So, we first hash then sign. But for that to be secure, it must be *impossible in practice* to exhibit two messages producing the same value after hashing. This is called a *collision*. So, we often require hash functions to be collision-resistant: it must be impossible to find x and y such that $H(x) = H(y)$. For this reason, $H(x)$ is often called the *digest*, or *fingerprint*, or *hash* of x.

We can define $\mathsf{Commit}(X; r) = (H(X\|r), X\|r)$ and $\mathsf{Open}(c, X\|r) = X$ if $H(X\|r) = c$ and fail otherwise, to obtain a commitment scheme. So, $H(X\|r)$ must hide X and also bind to X. In access control, we have also seen that $H(\mathsf{password})$ can be used to verify a password without disclosing it. So, we often require that given h, it is infeasible to find x such that $H(x) = h$. We say that H is *preimage-resistant*.

We can define a PRNG by $H(\mathsf{seed}\|\mathsf{counter})$. We can thus define a KDF function by

$$s \mapsto \mathsf{trunc}_\ell(H(s\|1)\|H(s\|2)\|\cdots)$$

to obtain an ℓ-bit key from a seed s.

The notion of preimage-resistance in hash functions has two variants: in the *first preimage resistance*, we require that an adversary who is given a digest h cannot find x such that $H(x) = h$. In the *second preimage resistance*, we require that an adversary who is given y cannot find $x \neq y$ such that $H(x) = H(y)$ (so, a second preimage makes a collision). Clearly, a bruteforce attack can do a preimage attack (a first preimage attack or a second preimage attack) with complexity $\mathcal{O}(N)$, where N is the size of the output range: we repeatedly pick a random (but new) x until $H(x)$ matches the target digest. Each selection produces a random digest, independent from the others. Assuming a uniform distribution, the probability to match the target is $\frac{1}{N}$. So, the expected number of iterations is

$$\sum_{i=1}^{+\infty} i\left(1 - \frac{1}{N}\right)^{i-1} \frac{1}{N} = N$$

The MD hash functions and follow ups. In 1990, Ronald Rivest proposed a hash function called MD4 [69]. ("MD" stands for "Message Digest".) It was quickly replaced by MD5, in 1991, which became a famous standard (RFC 1321 [70] in 1992). Both produce digests of 128 bits. A variant of MD4 and MD5 was proposed as a US standard in 1993 [1]. It was called SHA (Secure Hash Algorithm) but is now called SHA0 because it is became obsolete quickly after. It was replaced by SHA1 in 1995 [2]. SHA0 and SHA1 produce digests of 160 bits. Some new standard algorithms were proposed in 2002: the SHA2 family, which includes SHA256, SHA384, and SHA512 [3]. (These names indicate the bitlength of the produced digest.) The SHA3 standard appeared in 2015 [36]. So far, MD4, MD5, and SHA0 are badly broken [29, 33, 84, 86, 88]. There was a theoretical attack on SHA1 in 2005 [85]. Since then, a collision on SHA1 has been found [81]. Consequently, SHA1 is not recommended any more.

The general design of these hash functions consists of making a *compression function* from a kind of block cipher, then a hash function by iterating the compression on message blocks. To hash a message, the message is first padded, then split into blocks. Then, compression starts with an initial value and the first block, producing a chaining value. Each block is processed by compressing the previous chaining value and the block. The final chaining value is the digest.

Following the *Merkle-Damgård extension* [31, 59], the message X is transformed into a sequence of blocks X_i by

$$X_1\|\ldots\|X_n = X\|\mathsf{pad}(X)$$

where $\mathsf{pad}(X)$ consists of a bit 1 followed by a variable number of 0 bits and the encoding of the message length, so that the length of $X\|\mathsf{pad}(X)$ is multiple of the block length. It is also assumed that the padding is not larger than a block. Then, using a compression function C, we define $H_0 = \mathsf{IV}$, the initial value, and $H_i = C(H_{i-1}, X_i)$, $i = 1, \ldots, n$. Finally, H_n is the message digest. This construction comes with the following theorem:

Theorem 6.1. *If C is collision-resistant, then the constructed hash function is collision-resistant.*

To construct a compression function from a block cipher, we use the *Davies-Meyer construction* [91]

$$C(H, X) = \mathsf{Enc}_X(H) + H$$

where $+$ is a group operation, H is the chaining value, and X is the message block.

The SHA1 compression function [2] is similar:

Input: an initial hash a, b, c, d, e, a message block $x_0, \ldots, x_{15}$
Output: a hash a, b, c, d, e

$a_{\mathsf{initial}} \leftarrow a$
$b_{\mathsf{initial}} \leftarrow b$
$c_{\mathsf{initial}} \leftarrow c$
$d_{\mathsf{initial}} \leftarrow d$
for $i = 16$ to 79 **do**
  $x_i \leftarrow \mathsf{ROTL}^1\,(x_{i-3}\ \mathsf{XOR}\ x_{i-8}\ \mathsf{XOR}\ x_{i-14}\ \ \mathsf{XOR}\ x_{i-16})$
end for
for $i = 1$ to 4 **do**
  for $j = 0$ to 19 **do**
    $t \leftarrow \mathsf{ROTL}^5(a) + f_i(b, c, d) + e + x_{20(i-1)+j} + k_i$
    $e \leftarrow d$
    $d \leftarrow c$
    $c \leftarrow \mathsf{ROTL}^{30}(b)$
    $b \leftarrow a$
    $a \leftarrow t$
  end for
end for
$a \leftarrow a + a_{\mathsf{initial}}$
$b \leftarrow b + b_{\mathsf{initial}}$
$c \leftarrow c + c_{\mathsf{initial}}$
$d \leftarrow d + d_{\mathsf{initial}}$
$e \leftarrow e + e_{\mathsf{initial}}$

Here, k_i is a constant and the f_i functions are bitwise Boolean functions defined by

$$\begin{aligned}
f_1(b, c, d) &= \mathsf{if}\ b\ \mathsf{then}\ c\ \mathsf{else}\ d \\
f_2(b, c, d) &= b\ \mathsf{XOR}\ c\ \mathsf{XOR}\ d \\
f_3(b, c, d) &= \mathsf{majority}(b, c, d) \\
f_4(b, c, d) &= b\ \mathsf{XOR}\ c\ \mathsf{XOR}\ d
\end{aligned}$$

Note that

$$\begin{aligned}
\mathsf{if}\ x\ \mathsf{then}\ y\ \mathsf{else}\ z &= (x\ \mathsf{AND}\ y)\ \mathsf{OR}\ ((\mathsf{NOT}\ x)\ \mathsf{AND}\ z) \\
&= (x\ \mathsf{AND}\ y)\ \mathsf{XOR}\ ((\mathsf{NOT}\ x)\ \mathsf{AND}\ z) \\
\mathsf{majority}(x, y, z) &= (x\ \mathsf{AND}\ y)\ \mathsf{OR}\ (y\ \mathsf{AND}\ z)\ \mathsf{OR}\ (z\ \mathsf{AND}\ x) \\
&= (x\ \mathsf{AND}\ y)\ \mathsf{XOR}\ (y\ \mathsf{AND}\ z)\ \mathsf{XOR}\ (z\ \mathsf{AND}\ x)
\end{aligned}$$

SHA3. The SHA3 standard is no longer based on the MD family. It comes from the Keccak algorithm which is based on a *sponge construction* [19]. The standard appeared in 2015 [36]. Keccak was designed in Belgium by Bertoni, Daemen, Peeters, and Van Assche from STMicroelectronics and NXP Semiconductors.

Keccak is quite flexible as it includes several tunable parameters r, c, and d. However, SHA3 only kept four vectors of parameters, defining this way the four functions in the following table:

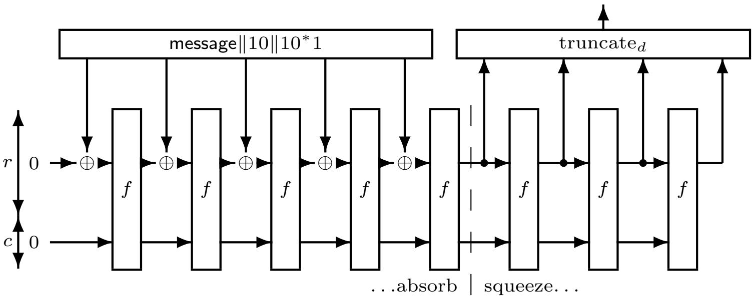

Figure 6.2: The Sponge Construction in SHA3

algo	r	c	d
SHA3-224	1 152	448	224
SHA3-256	1 088	512	256
SHA3-384	832	768	384
SHA3-512	576	1 024	512

The parameter d (giving the name of the function) is the bitlength of the output. The functions are based on a construction using a *state* of $b = r + c$ bits. This construction is called *sponge.* It consists of padding the input message, then processing it by chunks of r bits. The sponge processes each chunk of message through an *absorbing* phase. At each step, one chunk is XORed into the state and the state is updated using an invertible function f. Then, the sponge produce outputs through a *squeezing* phase. Each step produces r bits and it stops as soon as we reach d bits of output. This is illustrated on Fig. 6.2.

The function f (which is formally called Keccak-$f[b]$ in Keccak) operates on a state of b bits. This state is represented as a 3-dimensional array of $b = 5 \times 5 \times 2^\ell$ bits. The function is an iteration $n_r = 12 + 2\ell$ times of a round R defined by

$$R = \iota \circ \chi \circ \pi \circ \rho \circ \theta$$

All these functions are fixed and defined in the design. The operation θ is linear. The operations ρ and π only permute the bits in the state. The operation ι add constants (depending on the round index) modulo 2 to a few bits of the state. The operation χ is the only non-linear one. It is actually an invertible quadratic operation.

6.4 Message Authentication Codes

As already mentioned, a *message authentication code (MAC)* typically appends a *tag* to a message. This tag is computed based on a secret key and the message. The message is authenticated if it comes with a correct tag, based on the secret key. So, the primitive shall avoid that an adversary *forges* a message/tag pair not issued by the legitimate sender but still passing the authentication. Attacks scenario includes *key recovery* or just *forgery.* They can be run in a *known message attack*, in which the adversary collects messages which are randomly authenticated by the legitimate sender, or a *chosen message attack*, in which the adversary can select messages of his choice to be authenticated. (But, in the case of forgery attack, the forgery is valid if it was not authenticated by the legitimate sender.)

HMAC. One of the most popular MACs is the HMAC algorithm, which is a standard (RFC 2104 [52]). It is notably used in TLS and SSH.[1] It is based on a Merkle-Damgård hash function H. Roughly speaking, the tag of the message X with key K is computed by

$$\mathsf{HMAC}_K(X) = \mathsf{trunc}(H((K \oplus \mathsf{opad}) \| H((K \oplus \mathsf{ipad}) \| X)))$$

[1] See Section A.5.

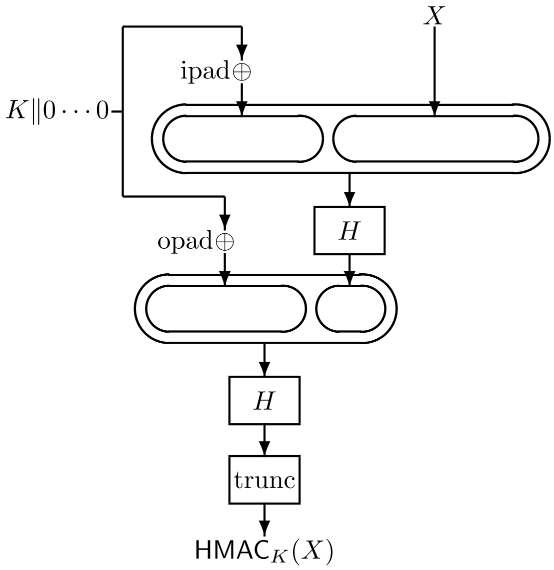

Figure 6.3: HMAC

where opad and ipad are constants defined by the standard. (See Fig. 6.3.) If the key K is too long (longer than a message block), it is first replaced by $H(K)$. Then and in any case, the key is concatenated with enough zero bytes so that the final length matches the one of a block.

In 2006, Bellare [14] proved that if the compression function defines a *pseudorandom function (PRF)* (in a sense to be defined in this chapter), then HMAC is also a PRF, which implies a MAC which is secure against existential forgeries. In the same year, Kim et al. [49] proved that HMAC based on several known hash functions (such as MD4 and SHA0) is not a PRF. In 2009, Wang et al. [87] have shown an attack on the PRF property of HMAC based on MD5 (with a quite high complexity though).

CBCMAC. Another popular construction (to be used with care) is based on a block cipher. The tag of a message (assumed to be a sequence of blocks) is the last ciphertext block of the CBC encryption of the message. This algorithm can be secure in some applications in two cases:

- the application makes sure that all messages have exactly the same length;
- the tag is only available to the adversary in some encrypted form.

Otherwise, CBCMAC alone is insecure as we can easily make forgeries. To see this, we can observe that if c is the CBCMAC of a message X_1, then the CBCMAC of $X_1\|B$, the concatenation of X_1 with a new block B, is the encryption $C_K(c \oplus B)$. Hence, if c' is the CBCMAC of a message X_2, with the knowledge of $X_1, X_2, B, c, c', C_K(c \oplus B)$, we can forge the CBCMAC of a new message $X_2\|B'$ for $B' = B \oplus (c \oplus c')$ as it is $C_K(c' \oplus B') = C_K(c \oplus B)$ (see Fig. 6.4).

There are several variants, including CMAC, which is the RFC 4493 standard [79].[2] RFC 4493 uses AES-128. So, we assume that the block length is of 128 bits and that the key has 128 bits. In CMAC, the message is split into blocks. The last block may be incomplete, in which case we pad it with the bitstring $10\cdots0$ to have a complete block. We obtain a sequence of blocks $x_1, \ldots, x_n$. We set $y_1 = C_K(x_1)$ where C_K is AES using the key K of CMAC. Then, we set $y_i = C_K(x_i \oplus y_{i-1})$ for $i = 2, \ldots, n-1$. Finally, $y_n = C_K(x_n \oplus y_{n-1} \oplus k_b)$, where $b \in \{1, 2\}$ and k_1 and k_2 are two values defined below. If x_n was not padded, we use $b = 1$. Otherwise, we use $b = 2$. We define $L = C_K(0)$, the encryption of the all-0 block. Now, we consider 128-bit blocks as elements of

[2] It used to be called OMAC.

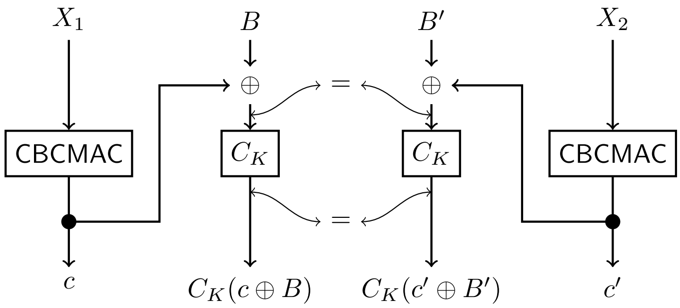

Figure 6.4: A Forgery Attack Against CBCMAC

$\mathsf{GF}(2^{128})$. For instance, $L = \sum_{i=1}^{128} L_i X^{128-i}$, where L_1 is the most significant bit of L and L_{128} is its least significant bit and X is a free variable. We set $k_1 = X \times L \bmod P(X)$ where

$$P(X) = X^{128} + X^7 + X^2 + X + 1$$

is the irreducible polynomial used to define the representation of $\mathsf{GF}(2^{128})$. We further set $k_2 = X \times k_1 \bmod P(X)$. This fully defines k_1 and k_2. The result of CMAC is y_n (or a truncation of it if we need a shorter tag than 128 bits). In 2003, Iwata and Kurosawa [45] (the original designers of CMAC) proved that if C is a pseudorandom permutation, then CMAC is secure.

PMAC. Another block-cipher based construction, which is not based on CBCMAC, was proposed by Black and Rogaway [20]. It is also proven secure if the block cipher is a pseudorandom permutation. It works by first computing the subkey L which is the encryption of the zero-block: $L = C_K(0)$. Then, the message is cut into blocks $x_1, \ldots, x_n$ (the last one could be incomplete but non-empty). We compute $\Sigma = \bigoplus_{i=i}^{n-1} C_K(x_i \oplus (2^i \cdot L)) \oplus x_n \oplus (2^n \cdot L)$, where $\cdot$ is the GF multiplication and 2 is a shorthand for the variable of the polynomial defining the GF structure.[3] If the last block is complete, Σ is replaced by $\Sigma \oplus (2^{-1} \cdot L)$. The MAC is then the first t bits of $C_K(\Sigma)$. The construction is depicted on Fig. 6.5.

WC-MAC. There is an analog to the Vernam cipher [83] for authentication which provides unconditional security. The construction was proposed by Wegman and Carter in 1981 [28, 89]. To authenticate a message X, we essentially encrypt a value $h_K(X)$ using the Vernam cipher, where h is an ε-XOR-universal hash function.[4]

Definition 6.2. *A family $(h_K)_K$ of functions is called a* ε-XOR-universal hash function *if for any a, x, y, with $x \neq y$, we have*

$$\Pr[h_K(x) \oplus h_K(y) = a] \leq \varepsilon$$

over the uniform selection of the random key K.

The construction WC-MAC is validated by the following theorem:

Theorem 6.3 (Krawczyk 1994 [51]). *If h is a ε-XOR-universal hash function, any chosen message attack against WC-MAC has a success probability bounded by ε.*

Proof. We consider an adversary making a chosen message forgery attack. Let d be the number of messages that the adversary made. We denote by $x_1, \ldots, x_d$ the messages queries by the adversary

[3] Be careful: this 2 is not $1 + 1$ in this structure!

[4] We shall not be confused by the name "hash function" as this notion has very little to do with the cryptographic hash functions which were previously discussed.

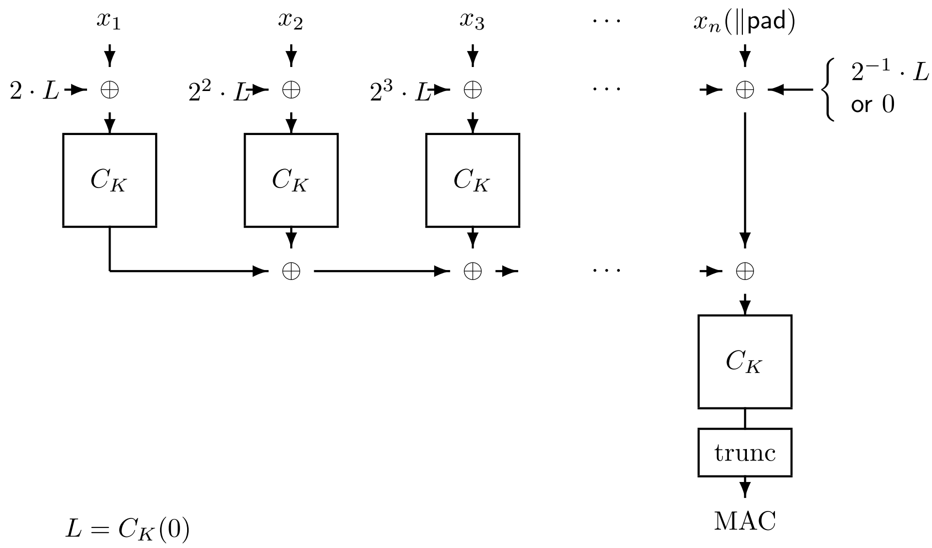

Figure 6.5: PMAC

and by $c_1, \ldots, c_d$ the corresponding authentication tags. We have $c_i = h_K(x_i) \oplus K_i$. Let (x, j, c) be the output of the adversary. It is a forgery if x is different from all x_i and if we have $c = h_K(x) \oplus K_j$.

In the first case, we have $j \notin \{1, \ldots, d\}$. So, K_j is uniform and independent from the entire experiment. Hence, $c = h_K(x) \oplus K_j$ occurs with probability exactly 2^{-m}, where m is the output bitlength of the hash function. We note that the sum over all a of $\Pr[h_K(x) \oplus h_K(y) = a]$ is equal to 1. Since the function is ε-XOR-universal, we deduce that $1 \leq 2^m \varepsilon$. So, this first case succeeds with probability bounded by ε.

In the second case, we have $1 \leq j \leq d$. The *view* of the adversary is characterized by the event that $c_i = h_K(x_i) \oplus K_i$ happens for all $i = 1, \ldots, d$. We let V denote this event. We let V' denote the same event for all i except $i = j$. We have

$$V = V' \cup [c_j = h_K(x_j) \oplus K_j]$$

We assume without loss of generality that the adversary is deterministic. So, x_i is a function of $c_1, \ldots, c_{i-1}$ and (x, j, c) is a function of $c_1, \ldots, c_d$. The probability of success in this case is $p = \sum_{c_1,\ldots,c_d} \Pr[c = h_K(x) \oplus K_j | V] \Pr[V]$. For each $(c_1, \ldots, c_d)$, we have

$$\begin{aligned} \Pr[c = h_K(x) \oplus K_j | V] &= \Pr[c = h_K(x) \oplus K_j | c_j = h_K(x_j) \oplus K_j] \\ &= \Pr[h_K(x) \oplus h_K(x_j) = c \oplus c_j | c_j = h_K(x_j) \oplus K_j] \end{aligned}$$

due to the independence between $(K_1, \ldots, K_{j-1}, K_{j+1}, \ldots, K_d)$ and (K, K_j). We use the Bayes formula to obtain

$$\begin{aligned} &\Pr[c = h_K(x) \oplus K_j | V] \\ &= \frac{\Pr[K_j = h_K(x_j) \oplus c_j | h_K(x) \oplus h_K(x_j) = c \oplus c_j]}{\Pr[K_j = h_K(x_j) \oplus c_j]} \times \Pr[h_K(x) \oplus h_K(x_j) = c \oplus c_j] \end{aligned}$$

and, thanks to the uniform distribution of K_j and its independence to K, we deduce

$$\Pr[c = h_K(x) \oplus K_j | V] = \Pr[h_K(x) \oplus h_K(x_j) = c \oplus c_j] \leq \varepsilon$$

This holds for all $(c_1, \ldots, c_d)$, so the probability of success in this case is bounded by ε multiplied by $\sum_{c_1,\ldots,c_d} \Pr[V] = 1$.

In both cases, the probability of success is bounded by ε, so the overall probability of success of the attack is bounded by ε. $\square$

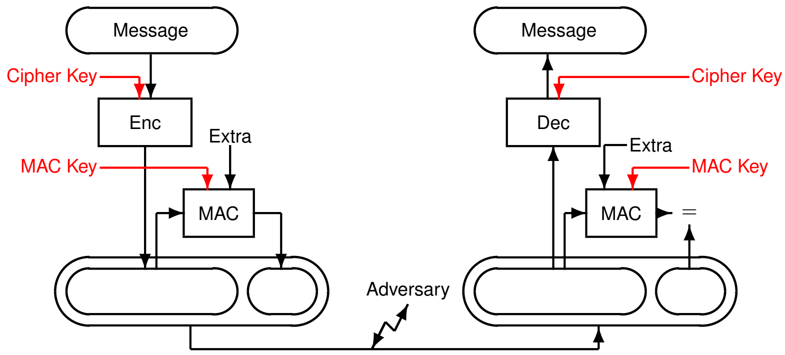

Figure 6.6: Encrypt-then-MAC

In 1994, Krawczyk proposed a ε-XOR-universal hash function based on an LFSR [51]: each key K defines an irreducible polynomial of degree m over $\mathbf{Z}_2$ and the initial state of the LFSR defined by this polynomial. Then, $h_K(x)$ is the XOR of the states of the LFSR at time t corresponding to a bit $x_t = 1$ in the message x.

In the GCM mode (see below), each key defines a Galois field element H, the message x defines a polynomial with binary coefficients, and $h_K(x)$ is simply the evaluation of this polynomial on H.

Just like the Vernam cipher can be replaced (at the cost of loosing the unconditional security) by a stream cipher, we can just encrypt $h_K(X)$ with a stream cipher as well. This may require synchronization or to transmit a nonce.

This type of construction has severe drawbacks. If, for one reason or another, the user uses a nonce value for two different messages X and X', it means that it applies the Vernam cipher with the same key to $h_K(X)$ and $h_K(X')$. Clearly, this leaks the value of $h_K(X) \oplus h_K(X')$. In many constructions, this could leak some information about K which allows to build forgeries. We will see it is the case for GCM.

Poly1305. One variant of the WC-mac is the Poly1305 one-time authenticator. It is used to authenticate a message using a key to be used only once. A key consists of a pair (r, s) of numbers between 0 and $2^{128} - 1$ (hence, 128-bit numbers). In applications, (r, s) is typically obtained by encrypting a nonce. A message to authenticate is a sequence of 128-bit numbers $m_1, \ldots, m_\ell$. The tag is computed by

$$(m_1 + 2^{128})r^\ell + \cdots + (m_\ell + 2^{128})r + s \bmod (2^{130} - 5)$$

$2^{130} - 5$ is a prime number.

Authenticated modes of operation. Authentication and encryption can be combined into an *authenticated encryption scheme*. There are lots of way to combine (in a "home-made manner") encryption schemes and authentication schemes. They can however badly fail.

The authenticated encryption techniques that we can use are the encrypt-then-MAC (see Fig. 6.6), MAC-then-encrypt (see Fig. 6.7), or the authenticated modes of operation such as CCM or GCM.

There exist some all-in-one mode of operation for block ciphers so that it integrates a MAC. These are *authenticated modes of operation*.

In the *CCM mode* (RFC 3610 [90]), the message is concatenated with its CBCMAC, then encrypted in CTR mode (CCM stands for "counter with CBCMAC"). More precisely, the algorithm is defined by two parameters $M \in \{4, \ldots, 16\}$ (the size of the tag in bytes) and $L \in \{2, \ldots, 8\}$ (the size of the length field in bytes). The encryption takes the key K, the message X (a stream of bytes), and a nonce N. The message is padded into $X \| \mathsf{pad}$ where pad consists of enough zero

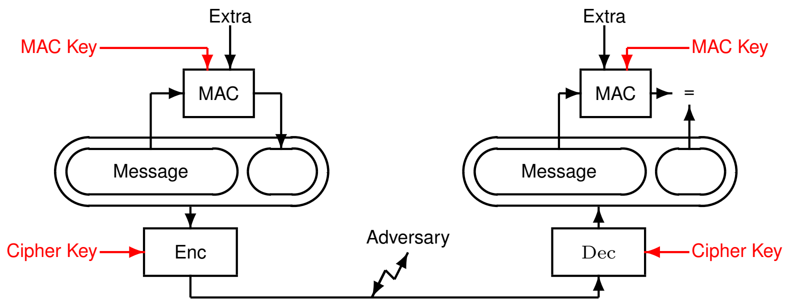

Figure 6.7: MAC-then-Encrypt

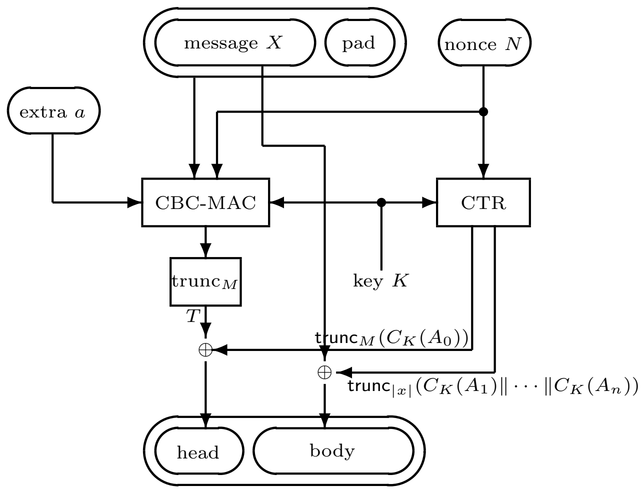

Figure 6.8: The Counter with CBCMAC (CCM) Authenticated Encryption Mode

bytes to complete the last block. Then, we split $X\|\mathsf{pad} = B_1\|\cdots\|B_n$ into blocks B_i. We define a byte byte_1 which encodes M and L. Then, we set $B_0 = \mathsf{byte}_1\|N\|\mathsf{length}(X)$ to define an initial block. Then, we compute the CBCMAC (with key K) of $B_0\|B_1\|\cdots\|B_n$ and truncate it to M bytes to obtain the tag T. I.e., we encrypt the block sequence in CBC mode with a zero initial vector and the last ciphertext block is truncated to define T. We define a byte byte_2 which encodes L. We define some blocks $A_i = \mathsf{byte}_2\|N\|i$ which will play the role of counters. Then, we encrypt T by $T \oplus \mathsf{trunc}_M(C_K(A_0))$ and X by $X \oplus \mathsf{trunc}_{\mathsf{length}(X)}(C_K(A_1)\|\cdots\|C_K(A_n))$ (see Fig. 6.8). The decryption is straightforward.

Interestingly, the CCM mode can also authenticate some *associated data* a at the same time as the encryption. For instance, secure messaging may like to bind a confidential message to the IP header in the protocol transporting the encrypting message. This information is not confidential but may be authenticated together with the message. For this, one special bit in byte_1 is flipped to indicate that a is used and a new sequence of blocks $\mathsf{length}(a)\|a\|\mathsf{pad}'$ is inserted between B_0 and B_1 in the CBCMAC computation. Then, the associated data a must be provided for the decryption.

In the *GCM mode* [35], the message is concatenated with its universal hash (see above), then encrypted in CTR mode. The encryption algorithm GCMAE takes a key K, some initial vector IV, the message X, and some associated data A. It returns a ciphertext C and a tag T. If there is no X, the algorithm is a MAC algorithm GMAC and the returned ciphertext is also empty. The decryption takes K, IV, C, T, and A, and returns X or an error message. The encryption is based

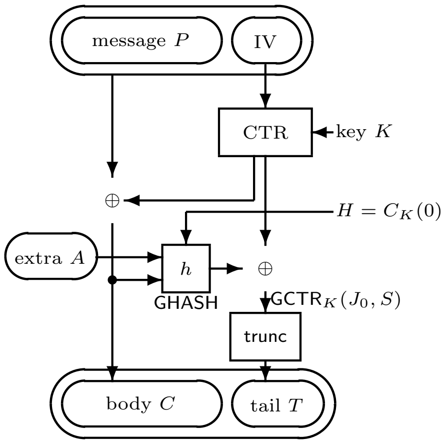

Figure 6.9: The Galois Counter Mode (GCM) Authenticated Encryption Mode

on GCTR, which encrypts X in CTR mode using a provided counter ct (see Fig. 6.9). I.e.,

$$\mathsf{GCTR}(\mathsf{ct}, X) = X \oplus \mathsf{trunc}_{\mathsf{length}(X)}(C_K(\mathsf{ct})\|C_K(\mathsf{ct}+1)\|\cdots)$$

We also define a universal hash function $\mathsf{GHASH}_H(X_1, \ldots, X_m)$ for a sequence of blocks $X_1, \ldots, X_m$ and a key H which is another block. Each block is taken as an element of $\mathsf{GF}(2^{128})$ and we define

$$\mathsf{GHASH}_H(X_1, \ldots, X_m) = X_1 H^m + \cdots + X_m H$$

in $\mathsf{GF}(2^{128})$. Then, we can define how GCMAE works: we set H to be the C_K-encryption of the all-zero block. We define a counter block $J_0 = \mathsf{IV}\|0^{31}1$ in binary (i.e., IV padded with 31 zero bits and a last bit set to 1). The ciphertext C is the GCTR-encryption of X with counter $J_0 + 1$. To compute T, we first determine the number u of zero bits to add to C to have an integral number of blocks (i.e., C padded with u zero bits has a length multiple of the block length), and the number v of zero bits to add to A to have an integral number of blocks. Then, we compute $S = \mathsf{GHASH}_H(A\|0^v\|C\|0^u\|\mathsf{length}(A)\|\mathsf{length}(C))$. Finally, we GCTR-encrypt S with counter J_0 and truncate the result into T.

To see the connection with the WC-MAC construction, we should see why GHASH is ε-XOR-universal. We can see that any message X defined a polynomial P_X whose degree is the block-length of X. We have $\mathsf{GHASH}_H(X) = P_X(H)$. Hence, $\mathsf{GHASH}_H(X) \oplus \mathsf{GHASH}_H(Y) = a$ is equivalent to $(P_X + P_Y)(H) = a$. When X and Y have a number of blocks bounded by m, this is a polynomial equation in H of degree bounded by m. Hence, it has no more than m roots. We deduce that the probability over H that $\mathsf{GHASH}_H(X) \oplus \mathsf{GHASH}_H(Y) = a$ holds is bounded by $m2^{-128}$.

Reusing a nonce in GCM is a disaster. If an adversary gets $T = \mathsf{GMAC}_K(\mathsf{IV}, A)$ and $T' = \mathsf{GMAC}_K(\mathsf{IV}, A')$ with $A \neq A'$, then $T \oplus T' = \mathsf{GHASH}_H(A) \oplus \mathsf{GHASH}_H(A')$ which is a polynomial equation with unknown H. We can solve it over the Galois field (the number of solutions we obtain is bounded by the size of A and A' in blocks, and we can later isolate the right solution). Once we know H, it is enough to authenticate anything. As we can XOR anything to a received ciphertext, we can easily forge the encryption of any message.

6.5 Formalism

We formalize a bit the notion of hash function.

Definition 6.4. *A **hash function** is a tuple $(\mathcal{D}, \{0,1\}^\tau, h)$ with a message domain $\mathcal{D} \subseteq \{0,1\}^*$,* *an output domain* $\{0,1\}^\tau$*, and one efficient deterministic algorithm* h *implementing a function*

$$\begin{array}{rccc} h: & \mathcal{D} & \longrightarrow & \{0,1\}^\tau \\ & X & \longmapsto & h(X) \end{array}$$

The notion of one-wayness (or resistance to the first preimage attack) is formalized as follows.

Definition 6.5. *A hash function* $(\mathcal{D},\{0,1\}^\tau,h)$ *is* (t,ε)-one-way *if for any probabilistic algorithm* $\mathcal{A}$ *limited to a time complexity and a code size of* t*, the advantage* Adv *is bounded by* ε*, where*

$$\mathsf{Adv} = \Pr[\textit{game returns } 1]$$

Game
1: $x \xleftarrow{\$} \mathcal{D}$
2: $y \leftarrow h(x)$
3: $\mathcal{A}(y) \rightarrow x'$
4: ***return*** $1_{x=x'}$

We could easily define the notion of resistance to the second preimage attack. However, it must be relative to the distribution of the first preimage which is selected as there is no notion of uniform distribution when $\mathcal{D}$ is infinite.

To define collision-resistance is more tricky. We could try to formalize it as follows.

Definition 6.6 (bad definition!). *A hash function* $(\mathcal{D},\{0,1\}^\tau,h)$ *is* (t,ε)-secure against collision attacks *if for any probabilistic algorithm* $\mathcal{A}$ *limited to a time complexity* t*, the advantage* Adv *is bounded by* ε*, where*

$$\mathsf{Adv} = \Pr[\textit{game returns } 1]$$

Game
1: $\mathcal{A}(y) \rightarrow x, x'$
2: ***return*** $1_{h(x)=h(x'),x\neq x'}$

However, no hash function would be secure following this definition. Indeed, whenever $\#\mathcal{D} > 2^\tau$, we know that a collision (x,x') exists. We could define an algorithm $\mathcal{A}$ which just prints this collision. This algorithm *exists* and works with very low complexity. Making a correct definition for collision-resistance is actually more difficult, and beyond the scope of this lecture.

We also formalize a bit the notion of message authentication code. For this, we limit ourselves to the most common construction of a symmetric message authentication scheme.

Definition 6.7. *A* ***message authentication code*** *is a tuple* $(\{0,1\}^k,\mathcal{D},\{0,1\}^\tau,\mathsf{MAC})$ *with a key domain* $\{0,1\}^k$*, a message domain* $\mathcal{D}\subseteq\{0,1\}^*$*, an output domain* $\{0,1\}^\tau$*, and one efficient deterministic algorithm* MAC *implementing a function from* $\{0,1\}^k \times \mathcal{D}$ *to* $\{0,1\}^\tau$*.*

Write $\mathsf{MAC}_K(\cdot) = \mathsf{MAC}(K,.)$*.*

So, k is the key length and τ is the tag (output) length. It is important that MAC is deterministic so that computing twice on the same input produce the same output. So, verifying if a tag is correct can be done by recomputing it.

Just like for symmetric encryption, we first define security against key recovery. We could consider *known message attacks* (KMA) or *chosen message attacks* (CMA). For simplicity, we formalize only CMA.

Definition 6.8. *A message authentication code* $(\{0,1\}^k,\mathcal{D},\{0,1\}^\tau,\mathsf{MAC})$ *is* (q,t,ε)-secure against key recovery under chosen message attacks *if for any probabilistic algorithm* $\mathcal{A}$ *limited to a time complexity* t *and to* q *queries, the advantage* Adv *is bounded by* ε*, where*

$$\mathsf{Adv} = \Pr[\textit{game returns } 1]$$

Game
$K \xleftarrow{\$} \{0,1\}^k$
$\mathcal{A}^{\mathsf{OMac}} \to K'$
return $1_{K=K'}$

Oracle $\mathsf{OMac}(X)$*:*
return $\mathsf{MAC}(K, X)$

A MAC always producing the same tag no matter the input is secure against key recovery (it does not use the key) but is clearly unusable to authenticate messages. So, we define the stronger notion of forgery attacks.

Definition 6.9. *A message authentication code* $(\{0,1\}^k, \mathcal{D}, \{0,1\}^\tau, \mathsf{MAC})$ *is* (q, t, ε)-secure against forgery under chosen message attacks *if for any probabilistic algorithm* $\mathcal{A}$ *limited to a time complexity* t *and to* q *queries, the advantage* Adv *is bounded by* ε*, where*

$$\mathsf{Adv} = \Pr[\textit{game returns } 1]$$

Game
$K \xleftarrow{\$} \{0,1\}^k$
$\mathsf{Queried} \leftarrow \emptyset$
$\mathcal{A}^{\mathsf{OMac}} \to (X, t)$
if $X \in \mathsf{Queried}$ ***then return*** *0*
return $1_{\mathsf{MAC}(K,X)=t}$

Oracle $\mathsf{OMac}(X)$*:*
$\mathsf{Queried} \leftarrow \mathsf{Queried} \cup \{X\}$
return $\mathsf{MAC}(K, X)$

Indeed, if $\mathcal{A}$ queries X to the authentication oracle, the oracle authenticates X, so we do not consider X as being forged.

If we have an algorithm $\mathcal{A}$ making a key recovery, we can transform it into an algorithm making a forgery as follows:

1: run $\mathcal{A} \to K$
2: pick a fresh X arbitrarily
3: compute $c = \mathsf{MAC}(K, X)$
4: return (X, c)

So, security against forgeries is stronger than security against key recovery.

Finally, we have an even stronger security notion which is the notion of *pseudorandom function.*

Definition 6.10. *A message authentication code* $(\{0,1\}^k, \mathcal{D}, \{0,1\}^\tau, \mathsf{MAC})$ *is a* (q, t, ε)-pseudorandom function (PRF) *if for any probabilistic algorithm* $\mathcal{A}$ *limited to a time complexity* t *and to* q *queries, the advantage* Adv *is bounded by* ε*, where*

$$\mathsf{Adv} = \Pr[\Gamma_1 \textit{ returns } 1] - \Pr[\Gamma_0 \textit{ returns } 1]$$

Game Γ_b
$K \xleftarrow{\$} \{0,1\}^k$
pick $F : \mathcal{D} \to \{0,1\}^\tau$
$\mathcal{A}^{\mathsf{O}} \to z$
return z

Oracle $\mathsf{O}(N, X)$*:*
if $b = 0$ ***then return*** $F(X)$
return $\mathsf{MAC}(K, X)$

Intuitively, an adversary who can make a forgery can check his forgery with an extra query. With the function $\mathsf{MAC}(K, .)$, it will say the forgery is correct. But with $F(\cdot)$, it will only confirm the forgery with probability $2^{-\tau}$ as F is random. So, if we can make a forgery, MAC cannot be pseudorandom.

More formally, given an adversary $\mathcal{A}$ making a forgery, we construct a distinguisher $\mathcal{D}$ as follows:

1: run $\mathcal{A}^{\mathcal{O}} \to (X, c)$
2: if X was queried by $\mathcal{A}$, output 0 and abort

3: query X to $\mathcal{O}$ and get c'
4: output $1_{c=c'}$

With the oracle $\mathcal{O} = \mathsf{MAC}(K, .)$, we have $\Pr[\mathcal{D}^{\mathsf{MAC}(K,.)} \to 1] = \Pr[\mathcal{A}\ \mathsf{wins}]$. With the oracle $\mathcal{O} = F(\cdot)$, we have $\Pr[\mathcal{D}^{F(\cdot)} \to 1] \leq 2^{-\tau}$. So,

$$\Pr[\mathcal{A}\ \mathsf{wins}] \leq \mathsf{Adv}(\mathcal{D}) + 2^{-\tau}$$

Where $\mathsf{Adv}(\mathcal{D}) = \Pr[\mathcal{D}^{\mathsf{MAC}(K,.)} \to 1] - \Pr[\mathcal{D}^{F(\cdot)} \to 1]$ is bounded by ε. Therefore, (q, t, ε)-PRF-security implies $(q - 1, t - t_0, \varepsilon + 2^{-\tau})$-unforgeability, where t_0 is the computational overhead of $\mathcal{D}$ compared to $\mathcal{A}$.

6.6 Bruteforce Collision Search Algorithms

If we pick in a set of N elements some independent and uniformly distributed samples, and if the number of samples is asymptotically $\theta\sqrt{N}$, the probability that at least one value is selected twice tends towards $1 - e^{-\frac{\theta^2}{2}}$. For instance, if $N = 365$ is the calendar and if we look at the birthday of random people in the calendar, we need less than 25 samples to find two persons with the same birthday with good probability. This may look paradoxical. This is actually called the *birthday paradox*. Nevertheless, the idea is that with $\mathcal{O}(\sqrt{N})$ we can find collisions within a constant probability.

Theorem 6.11. *Let $\theta > 0$ be a real number. If we pick n independent and uniformly distributed elements $X_1, \ldots, X_n$ in a set of cardinality N, if $n = o(N)$ as N goes to infinity, then the probability that at least two elements are equal is*

$$\Pr[\exists i < j\ X_i = X_j] = 1 - \frac{N!}{(N-n)!N^n} = 1 - e^{-\frac{n^2}{2N} + o(1)}$$

Before we give the formal proof, we provide here an informal computation which is easier to remember:

$$\begin{aligned} p &\approx 1 - \left(1 - \frac{1}{N}\right)^{\binom{n}{2}} \\ &\approx 1 - \left(1 - \frac{1}{N}\right)^{\frac{n^2}{2}} \\ &= 1 - e^{\frac{n^2}{2}\ln\left(1 - \frac{1}{N}\right)} \\ &\approx 1 - e^{-\frac{n^2}{2N}} \\ &= 1 - e^{-\frac{\theta^2}{2}} \end{aligned}$$

where we used that $\binom{n}{2} \approx \frac{n^2}{2}$ and $\ln\left(1 - \frac{1}{N}\right) \approx -\frac{1}{N}$.

Proof. Clearly, we have N^n possible sequences of n values. The number of sequences of pairwise different values is $\frac{N!}{(N-n)!}$. So, we have

$$\Pr[\exists i < j\ X_i = X_j] = 1 - \frac{N!}{(N-n)!N^n}$$

Now, we use the Stirling formula

$$m! \underset{m \to +\infty}{\sim} \sqrt{2\pi m}e^{-m}m^m$$

and obtain

$$\Pr[\exists i < j\ X_i = X_j] = 1 - \frac{N!}{(N-n)!N^n} \underset{N \to +\infty}{\sim} \left(1 - \frac{n}{N}\right)^{-N+n} e^{-n} = e^{-(N-n)\ln\left(1 - \frac{n}{N}\right) - n}$$

We now use $\ln(1-\varepsilon) = -\varepsilon - \frac{\varepsilon^2}{2} + o(\varepsilon^2)$ and obtain

$$\Pr[\exists i < j\ X_i = X_j] = 1 - \frac{N!}{(N-n)!N^n} \underset{N\to+\infty}{\sim} e^{-\frac{n^2}{2N}+o(1)}$$

□

This can be used to break hash functions: if we have digests of m bits, we take $N = 2^m$ and each random message x produces a random digest $h(x)$ in a set of size N. So, with $\theta 2^{\frac{m}{2}}$ trials, we find a collision with probability $1 - e^{-\frac{\theta^2}{2}}$.

In another variant of the above result, we can show that if we repeatedly pick samples until we find a collision, the expected number of samples before we stop with a collision is $\sqrt{\frac{\pi}{2}} \times \sqrt{N}$. In any case, it is $\mathcal{O}(\sqrt{N})$. The algorithm runs as follows:

Input: a cryptographic hash function h onto a domain of size N
Output: a pair (x, x') such that $x \neq x'$ and $h(x) = h(x')$

repeat
    pick a (new) random x
    compute $y = h(x)$
    insert (y, x) in the hash table
until there is already another (y, x') pair in the hash table
yield (x, x')

Note that this algorithm has a memory complexity of $\mathcal{O}(\sqrt{N})$ as well.

Theorem 6.12. *In the above algorithm, assuming that the obtained y values are independent and uniformly distributed, the expected number of iterations is asymptotically equivalent to* $\sqrt{\frac{\pi}{2}} \times \sqrt{N}$.

As an example, we can show an attack against a variant of CBCMAC called EMAC. Following EMAC, we use two keys K_1 and K_2. The MAC is the encryption under K_2 of the CBCMAC under K_1 of the message. Let us say that the block length is ℓ. To break EMAC, we submit $2^{\frac{\ell}{2}}$ chosen random message to get their tags and eventually get two messages $X_1 \neq X_2$ such that their tags t are equal. This means that their CBCMAC under key K_1 must be equal. We can now pick a random block B and ask for the tag t' of $X_1\|B$. Finally, the forged tag for $X_2\|B$ is also t'. This comes from t' being the $t' = C_{K_2}(B \oplus C^{-1}_{K_2}(t))$, due to the properties of the CBCMAC.

CBCMAC (and variants) are not the only constructions which are vulnerable to collisions at the birthday bound. We can also have a similar attack against PMAC. Indeed, assuming that 128 is the bitlength of blocks, we can get 2^{64} pairs $(B_i, \mathsf{PMAC}_K(x\|B_i))$ and 2^{64} pairs $(B'_j, \mathsf{PMAC}_K(x\|B'_j))$, where x is an arbitrary sequence of blocks, each B_i is an incomplete block, and each B'_i is a complete block. We expect to find a collision $\mathsf{PMAC}_K(x\|B_i) = \mathsf{PMAC}_K(x\|B'_j)$. Due to the structure of PMAC, we deduce (assuming $t = 128$) $C_K(B_i) = C_K(B'_j \oplus (2^{-1} \cdot L))$ so $2 \cdot (B_i \oplus B_j) = L$. With L, we can mount forgeries. For $t < 128$ we obtain candidates for L which can be filtered. Eventually, we recover the right L.

There exist also constant-memory algorithms to find collisions with complexity $\mathcal{O}(\sqrt{N})$. For instance, the Floyd cycle algorithm can be used [38]. The idea is that by picking x_0 at random and a permutation σ over the digest space, then iterating the function $F = \sigma \circ H$ on x_0 will eventually cycle. The graph of the obtained values will have a "ρ shape", with a tail and a loop. The point at which the tail connects to the loop is a collision. The magic is that, for this random function, the expected length of the tail λ and the expected length of the loop τ are both $\sqrt{\frac{\pi}{8}} \times \sqrt{N}$. So, the collision is found by visiting the graph in a clever way.

The Floyd algorithm is also called the tortoise and the hare algorithm. There are two animals (a tortoise and a hare) running over the trail with the form of a ρ, but one (the hare) goes twice faster than the other. They start from the tail of the ρ and will meet again inside the loop. The trick is that the number of steps until they meet again is necessarily a multiple of the length of the loop. So, by making and a new tortoise start from the tail as the other tortoise is also continuing, the two tortoises will meet at the collision point. The algorithm works as follows:

pick x_0, a permutation σ, and define $F = \sigma \circ H$

$a \leftarrow x_0$ (tortoise)
$b \leftarrow x_0$ (hare)
repeat
    $a \leftarrow F(a)$
    $b \leftarrow F(F(b))$
until $a = b$
$a \leftarrow x_0$
$a_{\mathsf{old}} \leftarrow \perp$
$b_{\mathsf{old}} \leftarrow \perp$
while $a \neq b$ **do**
    $a_{\mathsf{old}} \leftarrow a$
    $b_{\mathsf{old}} \leftarrow b$
    $a \leftarrow F(a)$
    $b \leftarrow F(b)$
end while
print a_{old} and b_{old}

The number i of iterations of the repeat loop is such that $\lambda \leq i \leq \lambda + \tau$. The number of iterations of the while loop is λ. So, the number of F computations is $3i + 2\lambda$. This is $\mathcal{O}(\sqrt{N})$.

As an example, we plotted the graph of a function over $\mathbf{Z}_{128}$ that we took arbitrarily by

$$x = uv \mapsto \text{first byte of SHA256("}\texttt{3.1415927-}uv\text{")} \bmod 128$$

where uv is x in hexadecimal. More precisely, the function was implemented by the following bash script:

```
#! /bin/bash

string="3.1415927"

for i in {0..127}
do
  j=`printf "$string-%02x" $i | sha256sum`
  j=`echo $j | tr "abcdef" "ABCDEF"`
  j=`echo $j | sed "s/^\(..\).*$/ibase=16;obase=2;\1/g" | bc`
  j=`echo $j | sed "s/.$//g"`
  j=`echo $j | sed "s/^/ibase=2;obase=A;/g" | bc`
  echo "$i -> $j"
done
```

To apply the previous result, we first note that $\sqrt{\frac{128\pi}{8}} \approx 7$. Hence, we should expect tails and loops to have length 7 on average. We plot the graph of the function on Fig. 6.10 and put in red all vertices which have a tail of at most 7.

6.7 How to Select Security Parameters?

Symmetric encryption must face the generic attacks of complexity 2^n, when n is the bitlength of the key. We take this as a reference for a security: a symmetric encryption scheme is secure if this is the best attack we can mount on it. So, the keylength is the security parameter. In general, we say that the bitlength-equivalent security is n if the best attack needs 2^n operations.

For hash functions with digests of n bits, as far as preimage attacks are concerned, the bitlength equivalent is n. But *if we care* for collisions, the bitlength equivalent is $\frac{n}{2}$. So, n shall be doubled compared to the key length in symmetric encryption to obtain a corresponding security.

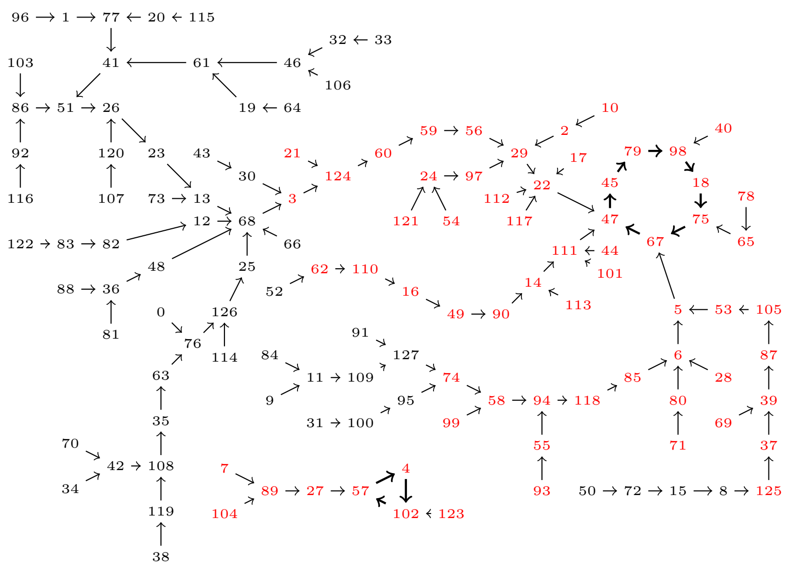

Figure 6.10: Graph of a Random Function

We should note that it is dangerous to underestimate the impact of academic attacks, specially about collisions. People think that "academic collisions" which are found are not so dangerous because they result after incredible efforts into colliding messages which have no meaning. It was however demonstrated that these attacks can be transformed into collision forgeries for media content with real meaning. Indeed, most of media format look like programming. There are several techniques to "hide" random-looking parts (which make the collision) into a content which makes sense. For instance, we can make a media document of three parts: a common prefix, a collision block, and a common suffix. The common prefix could be a program taking some data at a target address inside the collision block and interpreting it as another address, then the command to jump a this address.

Common prefix:

1: take addr at address target
2: jump to addr

If we build two (random-looking) collision blocks inside which this address becomes either addr_1 and addr_2, we can then build a common suffix which will have form

Common suffix:

1: [filler]
2: [starting at addr_1] content 1
3: [filler]
4: [starting at addr_2] content 2

The two documents formed this way will show either content 1 or content 2. Of course, the inner structure will show the trick, but the attack could be devastating.

6.8 Other Reasons why Security Collapses

Note that the security of symmetric encryption, MAC, and hashing, is often purely heuristic. We are not able to prove the security based on some well established hardness assumption like in public-key cryptography. So, we rather consider the primitives as secure until we have a proof that it is not the case. This assumes that discovered attacks become public. Assuming that the academic research is ahead of hidden research on cryptanalysis, this may be a reasonable assumption when the primitive gets enough exposure.

Security may collapse if the academic research discovers an attack on these primitives.

There is also a quantum threat on symmetric cryptography: Grover has shown that using quantum computer, the bitlength equivalent security is divided by two, compared to classical computers [43].

Symmetric primitives also suffer from side channel attacks: sometimes, implementation may leak some information, although the mathematical model of the primitive does not consider it.

It is also unfortunate that many security arguments in the literature happen to be incorrect. Sometimes, even security models are inappropriate, if not irrelevant at all.

Finally, the stand-alone security of primitives rarely offers any guaranty on the security when the primitive is used together with others in a complicated infrastructure.

This is the unfortunate current state of research in cryptography.

Chapter 7

Public-Key Cryptography

This chapter presents elementary notions of public-key cryptography such as public-key cryptosystems, digital signatures, and key agreement protocols. It focuses on RSA and ElGamal -like cryptography.

7.1 Public-Key Cryptography

In 1976, Diffie and Hellman published the seminal paper "New Directions in Cryptography". This gave birth to public-key cryptography [32]. In this paper, they proposed the notions of *trapdoor permutation*, *public-key cryptosystem*, *digital signature scheme*, and *key agreement protocol*. They provided an instance only for the last primitive.

A trapdoor permutation is defined by a probabilistic algorithm Gen and two deterministic algorithms Perm and $\mathsf{InvPerm}$. Gen generates a pair (param, K) (in which K is called the *trapdoor*) such that $\mathsf{Perm}_{\mathsf{param}}$ and $\mathsf{InvPerm}_K$ define two permutations over a given domain which are inverse of each other. So, for any input X in the domain, we have $\mathsf{InvPerm}_K(\mathsf{Perm}_{\mathsf{param}}(X)) = X$. The security notion relates to the secrecy of X even though $\mathsf{Perm}_{\mathsf{param}}(X)$ and param are public.

In a public-key cryptosystem, this is essentially the same with different notations, without requiring $\mathsf{Perm}_{\mathsf{param}}$ to be a permutation or even deterministic: we have two probabilistic algorithms Gen and Enc and one deterministic one Dec. Gen generates a pair $(\mathsf{pk}, \mathsf{sk})$ where pk is called the *public key* and sk is called the *secret key*, and such that for any X in the domain, we always have $\mathsf{Dec}_{\mathsf{sk}}(\mathsf{Enc}_{\mathsf{pk}}(X)) = X$. The security notion relates to the secrecy of X even though $\mathsf{Enc}_{\mathsf{pk}}(X)$ and pk are public.

In a key agreement protocol (also called *key exchange* or *key establishment*), there are two probabilistic interactive algorithms with no input which generate the same output (the key) when interacting with each other. It should be such that this output is secret even though the messages between the two algorithms are public.

Public-key cryptosystems are the asymmetric equivalent to symmetric encryption. There is also an asymmetric equivalent to MAC which is the called a *digital signature scheme*. In a digital signature scheme, we have two probabilistic algorithms Gen and Sig and one deterministic one Ver. Gen generates a pair $(\mathsf{pk}, \mathsf{sk})$ where pk is called the *public key* and sk is called the *secret key*, and such that for any X in the domain, we have $\mathsf{Ver}_{\mathsf{pk}}(\mathsf{Sig}_{\mathsf{sk}}(X)) = X$ for sure. The security notion relates to the unforgeability of a valid signature Y (i.e. a value such that $\mathsf{Ver}_{\mathsf{pk}}(Y)$ does not abort) even though pk is public.

A digital signature scheme is typically used in a *certificate*: a server holding a public key K requests a *certificate authority* to sign (by $\mathsf{Sig}_{\mathsf{sk}}$) a message saying that K belongs to the server. The signature is verified by $\mathsf{Ver}_{\mathsf{pk}}$. So, a client holding pk can verify the certificate and extract K from it. Based on K, he can establish a secure communication with the server.

7.2 Diffie-Hellman Key Exchange

We already covered the Diffie-Hellman key exchange protocol in Chapter 2. Essentially, the two algorithms (one for Alice and one for Bob) exchange their public keys and derive a key from their secret key and the received public key. The public key comes with a secret one. In *static* mode, the public keys are long-term values and may be obtained from a directory. In *ephemeral* mode, the public keys are freshly generated for each session of the protocol. In *semi-static* mode, one key is fresh and the other is a long-term one.

In ephemeral mode, the generated secret key can be erased after the protocol completes since the next session will generate a new one. This has the nice property of adding *forward secrecy*: If any long-term key is corrupted in the far future, this cannot compromise the secrecy of encryptions which are done now using the output of the key exchange. Indeed, there is no longer any long-term secret. In static mode, the corruption of the state of Alice, for instance, makes possible to recover the output of the protocol (by feeding Alice's algorithm with her state and Bob's public key). So, secrecy is not guaranteed in that case. Forward secrecy also appears in the Signal protocol, which is discussed in Section A.4.

Diffie and Hellman did not propose any public-key cryptosystem. They could have transformed their key exchange protocol in a hybrid encryption model, by using the output of the protocol with symmetric encryption or by proposing the ElGamal cryptosystem (which appeared in 1984 [37]). Actually, the first proposal of a secure cryptosystem was RSA [71].

7.3 RSA Cryptography

In 1978, Rivest, Shamir, and Adleman proposed their RSA cryptosystem [71]. It is actually a trapdoor permutation. It can also be used in a reversed way as a digital signature scheme. We have already seen it. Actually, what we saw is often called *plain RSA*, *textbook RSA*, *vanilla RSA*, or even *raw RSA*, as it is not used as a real-life cryptosystem. It is the one we find in textbooks or low-level introduction to cryptography. Indeed, real-life messages are not $\mathbf{Z}_N$ numbers. Furthermore, RSA has some homomorphic properties such as $\mathsf{Enc}(ab) = \mathsf{Enc}(a)\mathsf{Enc}(b)$, which may lead to potential weaknesses. Different ways to implement RSA may also leak some information.

A popular standard (although outdated) is PKCS#1v1.5 [8]. Essentially, to encrypt a message M, we run the textbook RSA on the byte string $00\|02\|\mathsf{PS}\|00\|M$ converted into an integer, where PS is a string of random nonzero bytes such that the complete byte string has the same length as the modulus N. To decrypt, we check that the textbook-RSA decryption parses to this type of byte string and extract M. The size of the message to encrypt is limited so that PS has at least 64 bits.

This standard is supposed to be replaced by RSA-OAEP [9]. There, we apply the textbook RSA to the string $00\|\mathsf{maskedSeed}\|\mathsf{maskedDB}$, where $\mathsf{maskedDB} = \mathsf{DB} \oplus \mathsf{MGF}(\mathsf{seed})$, $\mathsf{maskedSeed} = \mathsf{seed} \oplus \mathsf{MGF}(\mathsf{maskedDB})$, seed is selected at random, and DB is the concatenation of some padding string with the message M (see Fig. 7.1). MGF is an ad-hoc function in the PKCS#1 standard based on a hash function. Decryption is straightforward.

We can construct an equivalent cryptosystem to RSA which uses $e = 2$ (this is not allowed in RSA since 2 is never coprime with $\varphi(N)$). This is called the *Rabin cryptosystem* [66] (see Fig. 7.2). The idea is that we have to extract a square root to decrypt. Unfortunately, decryption is ambiguous since we have four square roots. One way to get around this problem is to add enough redundancy in the message to encrypt so that it is unlikely that there will be more than one square root passing the redundancy check. For instance, the redundancy may consists of having 64 specific bits set to zero. One concrete proposal is the SAEP-Rabin cryptosystem [21] in which the encryption is the textbook-Rabin encryption applied on $\mathsf{maskedDB}\|\mathsf{seed}$, with $\mathsf{maskedDB} = \mathsf{DB} \oplus \mathsf{MGF}(\mathsf{seed})$ and $\mathsf{DB} = M\|00\cdots0$ with a fixed (but large enough) number of 0 bits in the padding.

We say that a digital signature scheme has the property of *message recovery* if from $\sigma = \mathsf{Sig}_{\mathsf{sk}}(X)$, we can extract X by $\mathsf{Ver}_{\mathsf{pk}}(\sigma)$. This corresponds to the way we defined signature schemes,

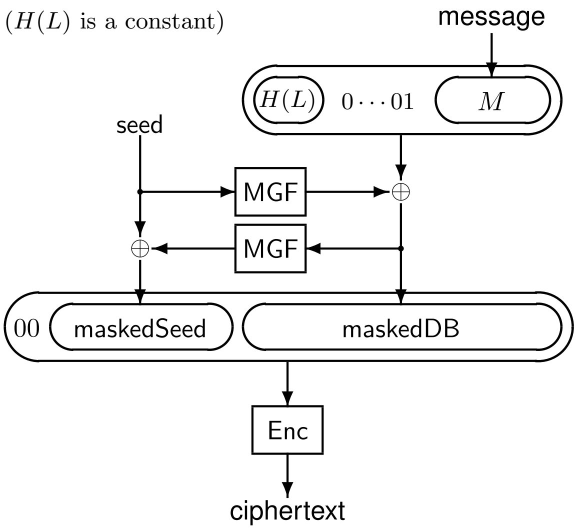

Figure 7.1: OAEP Formatting for RSA Encryption

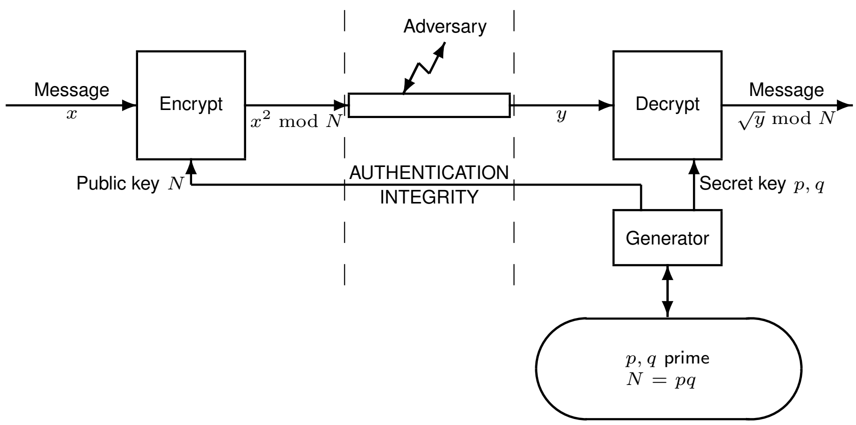

Figure 7.2: The Rabin Cryptosystem

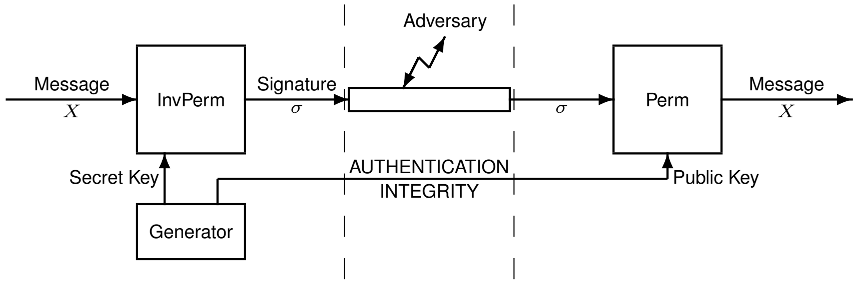

Figure 7.3: Signature with Message Recovery from Trapdoor Permutation

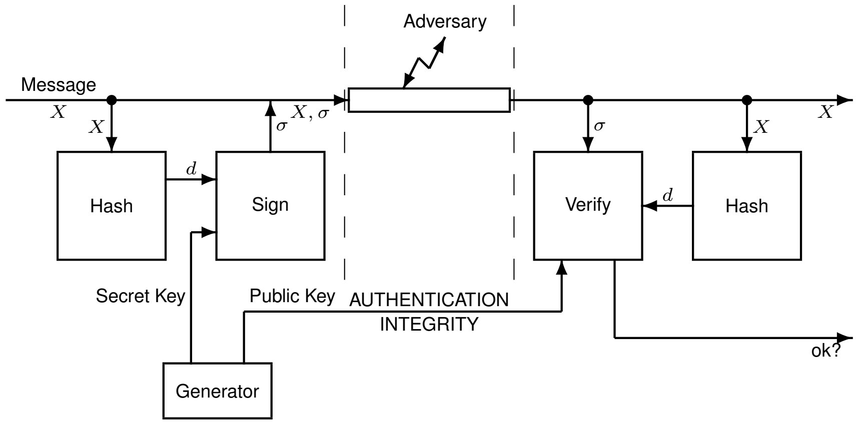

Figure 7.4: The Hash-and-Sign Paradigm

In many concrete schemes, the output of $\mathsf{Sig}_{\mathsf{sk}}(X)$ is of form $X\|\sigma$ where σ is called the *signature*. In that case, $\mathsf{Ver}_{\mathsf{pk}}(X\|\sigma)$ is aborting if σ is an invalid signature of X, and producing X as an output otherwise. Quite often, we say that that σ (instead of $X\|\sigma$) is the output of $\mathsf{Sig}_{\mathsf{sk}}$ and that it must be concatenated to X for transmission. We then say that the signature has no message recovery as the verification needs both X and σ instead of σ alone.

Using a trapdoor permutation in a reversed way, we can construct a signature scheme with message recovery: we just say that $\mathsf{Sig}_{\mathsf{sk}}(X)$ is the decryption of X using $\mathsf{InvPerm}$ and that $\mathsf{Ver}_{\mathsf{pk}}(\sigma)$ is just running Perm (see Fig. 7.3). When Perm and $\mathsf{InvPerm}$ are the RSA encryption and decryption algorithms, this is the textbook-RSA signature scheme. So, textbook-RSA can be regarded as a digital signature scheme with message recovery.

A more popular way to construct a signature scheme (without message recovery) from a trapdoor permutation is by using a hash function. In this construction, we have $\mathsf{Sig}(X) = \mathsf{InvPerm}(H(X))$. To run $\mathsf{Ver}(X, \sigma)$, we compare $H(X)$ with $\mathsf{Perm}(\sigma)$ and yield X if they match. It can be generalized to extend the domain of any elementary signature scheme Sig^0 by $\mathsf{Sig}(X) = \mathsf{Sig}^0(H(X))$ (see Fig. 7.4). This type of construction is actually called the *hash-and-sign* paradigm.

PKCS#1v1.5 also includes a signature scheme: we apply the textbook-RSA signature (see Fig. 7.5) on the byte string $00\|01\|\mathsf{FF}\cdots\mathsf{FF}|00\|D$ where D is the encoding (following a specific format) of H and $H(X)$. Again, the padded string must have the same length as the modulus. To verify a signature σ of a message X, we apply the textbook-RSA message recovery based on

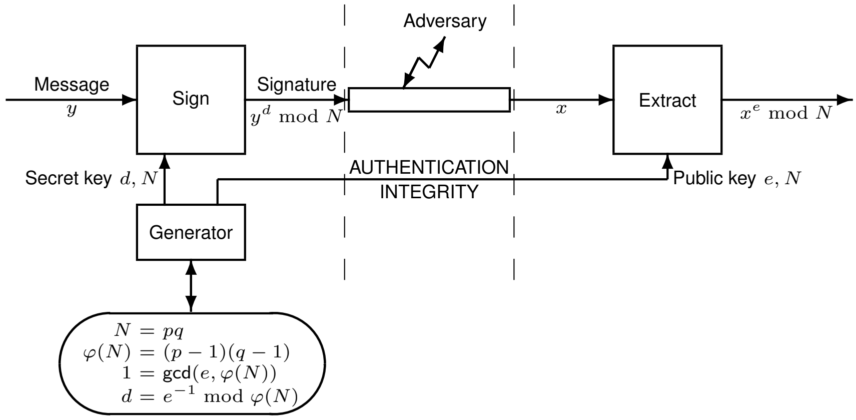

Figure 7.5: RSA Signature

σ, check that it parses into the correct format, extract H and digest from D, then compare $H(X)$ with digest. Interestingly, the choice of H is not fixed as it is specified in D. There is a list of allowed hash functions though. E.g., SHA1.

This outdated standard is supposed to be replaced by *RSA-PSS* [9], where the textbook-RSA signature is applied on $\mathsf{maskedDB}\|\mathsf{H}\|\mathsf{BC}$, where BC is an hexadecimal constant byte, $\mathsf{maskedDB} = 0\cdots 0\|\mathsf{salt} \oplus \mathsf{MGF}(\mathsf{H})$, and $\mathsf{H} = H(0\cdots 0\|H(X)\|\mathsf{salt})$ (see Fig. 7.6).

7.4 ElGamal Cryptography

The ElGamal cryptosystem [37], which was already covered in Chapter 2, comes with an ElGamal signature scheme which is quite independent. In the group $\mathbf{Z}_p^*$ generated by some public g, where p is a public prime number, the secret key is some x (modulo $p-1$) and the public key is $y = g^x \bmod p$. To sign M, we pick a random $k \in \mathbf{Z}_{p-1}^*$ and compute $\sigma = (r,s)$, where $r = g^k \bmod p$ and $s = \frac{H(M)-xr}{k} \bmod (p-1)$ (see Fig. 7.7). A signature (r,s) is valid for M if $0 \le r < p$ and $y^r r^s \equiv g^{H(M)} \pmod{p-1}$.

This scheme is not favored, because the signatures are quite long (we need two number of the size of p, which can be pretty large) and there is a lack of uniform security proof. Indeed, we can construct some parameters p and g making the signature insecure. Nevertheless, this scheme initiated a long dynasty of variants.

We must mention an important problem in the ElGamal signatures: although it is required that k is picked at random for every new signature generation, it could be the case that, occasionally, some k repeats. The consequences of a k repetition are here terrible. Indeed, from two signed messages (M_1, r_1, s_1) and (M_2, r_2, s_2) using the same value k (this is visible from $r_1 = r_2$), we deduce that

$$\frac{s_1}{s_2} \equiv \frac{H(M_1) - xr_1}{H(M_2) - xr_2} \pmod{p-1}$$

which allows to deduce x. Hence, the secret key leaks from a single repetition of k!

In 1989, Schnorr [73, 74] introduced a second prime number q, factor of $p-1$, and worked in a subgroup of $\mathbf{Z}_p^*$ of order q. The scheme was also changed so that the signature consisted of a digest and a number of the size of q.

In 1995, DSA was adopted as a US signature standard (the version 2 is available as reference [4]). It was essentially based on the ideas by Schnorr with a slightly changed algorithm.

Some other variants followed: Nyberg-Rueppel in 1995 [61], Pointcheval-Vaudenay [27] in 1997, KCDSA in 1998 [57], ... Quite importantly, ECDSA, the elliptic-curve variant of DSA, appeared in

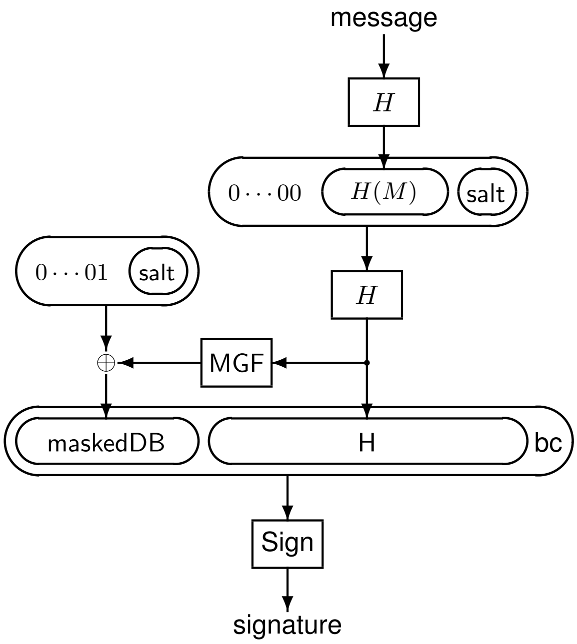

Figure 7.6: PSS Formatting for RSA Signature

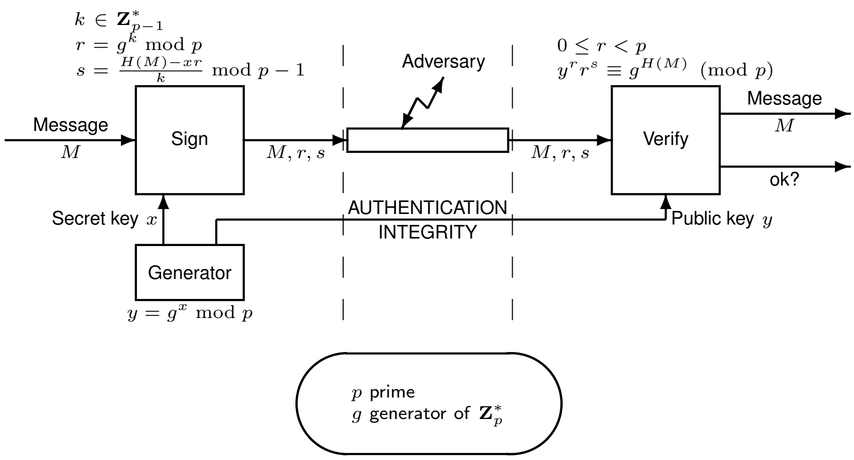

Figure 7.7: ElGamal Signature

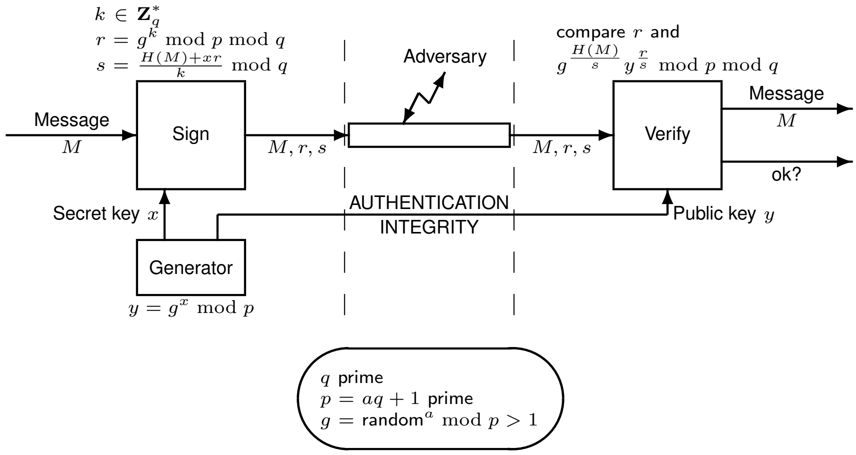

Figure 7.8: DSA Signature

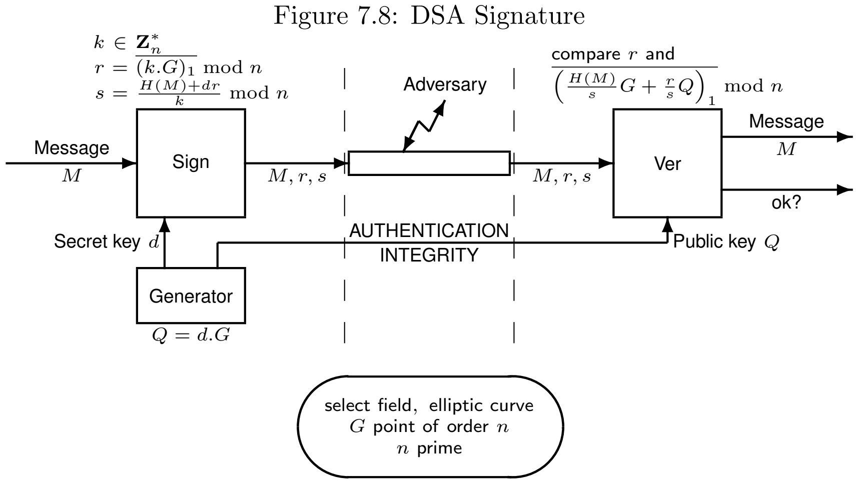

Figure 7.9: ECDSA Signature

1998 [4]. The main idea was to work in a group defined by an elliptic curve instead of a subgroup of $\mathbf{Z}_p^*$.

To generate some parameters for the above variants (except ECDSA), we first select a prime number q, then take $p = aq+1$ for a random a, until p is prime. Then a random element of $\mathbf{Z}_p^*$ is raised to the power a to get g until g if different from 1.

In DSA, the secret key is some $x \in \mathbf{Z}_q$. The public key is $y = g^x \bmod p$. To sign M, we pick $k \in \mathbf{Z}_q^*$ and compute $\sigma = (r, s)$, with $r = (g^k \bmod p) \bmod q$ and $s = \frac{H(M)+xr}{k} \bmod q$ (see Fig. 7.8). A signature (r, s) for M is valid if $r = \left(g^{\frac{H(M)}{s}} y^{\frac{r}{s}} \bmod p\right) \bmod q$. (Note that the fractions in the exponents are taken modulo q.)

For ECDSA, the public parameters consist of some finite field and some elliptic curve over this field, together with a reference point G generating a group of order n in the elliptic curve. The number n is also part of the public parameters and must be prime. A secret key is a number $d \in \mathbf{Z}_n^*$. A public key is a point $Q = dG$. To sign M, we pick $k \in \mathbf{Z}_n^*$ and compute the point kG. It has two coordinates x_1 and y_1 which are field elements. The element x_1 is converted into an integer $\bar{x}_1 \in \mathbf{Z}_n$ following a standard scheme. Then, we compute $r = \bar{x}_1 \bmod n$ and $s = \frac{H(M)+dr}{k} \bmod n$. If either r or s is zero, the algorithm shall restart. The signature is $\sigma = (r, s)$ (see Fig. 7.9). A signature (r, s) for M is valid if both r and s are in $\mathbf{Z}_n^*$, Q is a valid public key, and $r = \bar{x}_1 \bmod n$ where x_1 is the first coordinate of the point $u_1G + u_2Q$, with $u_1 = \frac{H(M)}{s} \bmod n$ and $u_2 = \frac{r}{s} \bmod n$.

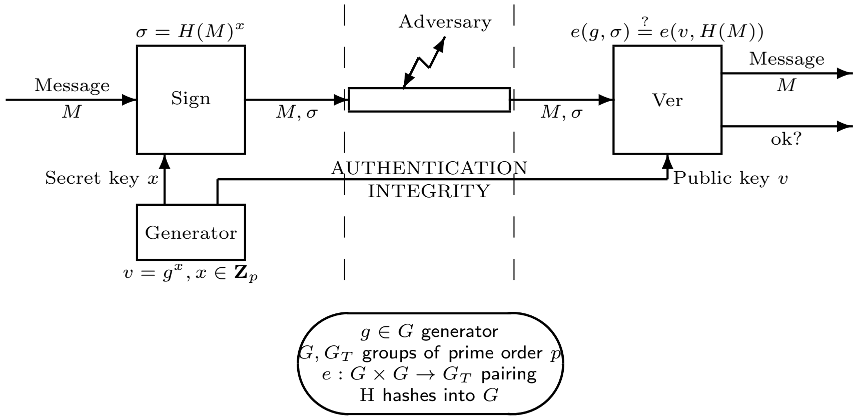

Figure 7.10: BLS Signature

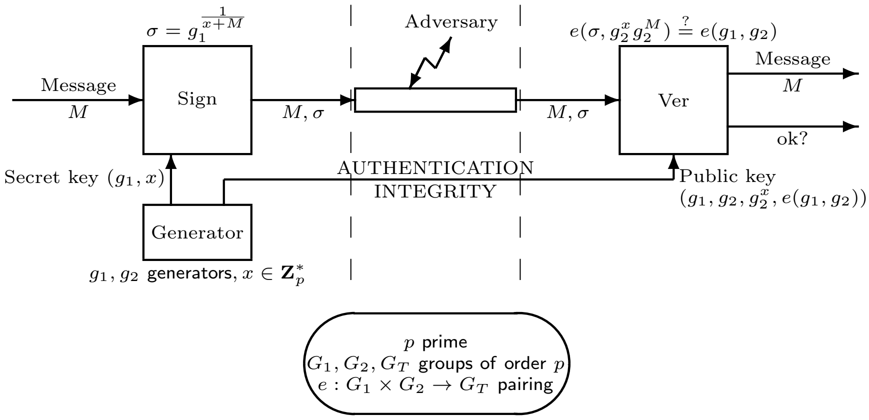

Figure 7.11: Simplified Boneh-Boyen Signature

The BLS signature scheme [25] (Boneh-Lynn-Shacham) which is based on pairings allow to have a signature which is a single group element σ. Here, this is one point on an elliptic curve. Using point compression, this is a single finite field element. Hence, the signature is pretty short compared to other schemes. The signature scheme is depicted on Fig. 7.10.

The Boneh-Boyen signature scheme [22, 23] is also based on pairing. What is new is that is requires no hashing for security. (Previous constructions all require a function H to behave like a random function to be secure.) Here, we can sign a finite field element. A simplified version of the signature scheme is depicted on Fig. 7.11.

7.5 Selecting Key Lengths

So far, the best way to break RSA is to factor N. So, the keys must be chosen such that factoring N is infeasible. We just have to adjust the length of p and q and avoid some known forms of weak moduli.

For schemes such as Diffie-Hellman, ElGamal and the variants, the best way is to solve the discrete logarithm problem. So, the parameters shall just make this infeasible. For this, we favor groups of prime order. Either we work in a subgroup of $\mathbf{Z}_p^*$, given a prime p (in which case we must specify the size of p and the size of the order), or we work over an elliptic curve, in which case we must select a finite field and the size of the order of the curve.

We compare the security with the cost of bruteforce attack on symmetric encryption. There exist several tables proposing vector of equivalent security parameters. The order of magnitudes are very similar. For instance, a table by Lenstra [55] suggests that the following security parameters propose equivalent security:

- symmetric encryption with a 82-bit key;
- RSA with a 1613-bit modulus;
- discrete logarithm with a subgroup of order q of $\mathbf{Z}_p^*$, where p has 1613 bits and q has 145 bits;
- an elliptic curve over a field whose cardinality has 154 bits;
- a hash function with digest length of 163 bits.

Those parameters are not considered as being enough for security nowadays. We recall that the digest length is doubled compared to the key length in symmetric encryption, due to the birthday paradox. We can see that the order of the groups on which we operate must have a similar size, due to a generic attack of square root complexity, but the p's and RSA moduli must be much larger.

7.6 Formalism

We first define a public-key cryptosystem.

Definition 7.1. *A* public-key cryptosystem *is a tuple* $(\mathsf{Gen}, \mathcal{M}, \mathsf{Enc}, \mathsf{Dec})$ *with a plaintext domain* $\mathcal{M} \subseteq \{0,1\}^*$ *and three efficient algorithms* Gen*,* Enc*, and* Dec*. The algorithm* Dec *is deterministic and output either something in* $\mathcal{M}$ *or an error* $\perp$*. It is such that*

$$\forall X \in \mathcal{M} \quad \Pr[\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, X)) = X] = 1$$

where $(\mathsf{pk}, \mathsf{sk})$ *is generated from running* Gen*. The probability is over the randomness used in* Gen *and* Enc*.*

We stress that decryption must be deterministic while encryption may be probabilistic. Indeed, a single plaintext may have several possible ciphertexts but they must all decrypt to the right plaintext. Next, we define IND-CPA and IND-CCA security as follows.

Definition 7.2. *A PKC* $(\mathsf{Gen}, \mathcal{M}, \mathsf{Enc}, \mathsf{Dec})$ *is* (t, ε)-secure under chosen plaintext attacks *(IND-CPA-secure) if for any interactive process* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *limited to a time complexity* t*, given a bit* b*, when we first run the following steps*

Game Γ_b*:*
1: $\mathsf{Gen} \to (\mathsf{pk}, \mathsf{sk})$
2: $\mathcal{A}_1(\mathsf{pk}) \to (\mathsf{pt}_0, \mathsf{pt}_1, \mathsf{st})$
3: ***if*** $|\mathsf{pt}_0| \neq |\mathsf{pt}_1|$ ***then return*** *0*
4: $\mathsf{Enc}(\mathsf{pt}_b) \xrightarrow{\$} \mathsf{ct}$
5: $\mathcal{A}_2(\mathsf{st}, \mathsf{ct}) \to z$
6: ***return*** z

we have

$$\Pr[\Gamma_1 \text{ returns } 1] - \Pr[\Gamma_0 \text{ returns } 1] \leq \varepsilon$$

It is (q, t, ε)*-**secure under chosen plaintext/ciphertext attacks** (IND-CCA-secure) if the same holds for any similar interactive process* $\mathcal{A}^{\mathsf{Dec}(\mathsf{sk},.)}$ *who is limited to* q *queries to a decryption oracle* $\mathsf{Dec}(\mathsf{sk}, .)$ *but not allowed to send it* c*.*

So, we have a game in which the adversary proposes two messages. Then, one of the two is encrypted and the adversary must guess which one of the two from the ciphertext. Clearly, if encryption is deterministic, this is easy since the adversary can just encrypt the messages himself and compare with the challenge ciphertext. So, the RSA cryptosystem that we saw is not IND-CPA secure. Most modern cryptosystems are actually probabilistic. Some are just variants of the ElGamal cryptosystem which starts by making a key agreement on a random ephemeral key then use the generalized Vernam cipher with this key.

The restriction that m_0 and m_1 must have the same length comes from the impossibility to make a secure cryptosystem on $\{0,1\}^*$: it is actually impossible to perfectly hide the message length. So, we must live with cryptosystems leaking the message length.

We define digital signature schemes as follows.

Definition 7.3. *A* digital signature scheme *is a tuple* $(\mathsf{Gen}, \mathcal{D}, \mathsf{Sig}, \mathsf{Ver})$ *with a message domain* $\mathcal{D} \subseteq \{0,1\}^*$ *and three* *efficient* *algorithms* Gen*,* Sig*, and* Ver*. The algorithm* Ver *is deterministic and outputs 0 (reject) or 1 (accept). It is such that*

$$\forall X \in \mathcal{D} \quad \Pr[\mathsf{Ver}(\mathsf{pk}, X, \mathsf{Sig}(\mathsf{sk}, X)) = 1] = 1$$

where $(\mathsf{pk}, \mathsf{sk})$ *is generated from running* Gen*. The probability is over the randomness used in* Gen *and* Sig*.*

Definition 7.4. *A digital signature scheme* $(\mathsf{Gen}, \mathcal{D}, \mathsf{Sig}, \mathsf{Ver})$ *is* (q, t, ε)*-**secure against existential forgery under chosen message attacks** (EF-CMA) if for any probabilistic algorithm* $\mathcal{A}$ *limited to a time complexity* t *and to* q *queries, the advantage* Adv *is bounded by* ε*.*

$$\mathsf{Adv} = \Pr[\textit{game returns } 1]$$

Game
1: $\mathsf{Gen} \xrightarrow{\$} (\mathsf{pk}, \mathsf{sk})$
2: $\mathsf{Queries} \leftarrow \emptyset$
3: $\mathcal{A}^{\mathsf{OSig}}(\mathsf{pk}) \to (X, \sigma)$
4: ***if*** $X \in \mathsf{Queries}$ ***then return*** *0*
5: ***return*** $1_{\mathsf{Ver}(\mathsf{pk}, X, \sigma)}$

Oracle $\mathsf{OSig}(X)$*:*
6: $\sigma \leftarrow \mathsf{Sig}(\mathsf{sk}, X)$
7: $\mathsf{Queries} \leftarrow \mathsf{Queries} \cup \{X\}$
8: ***return*** σ

So, EF-CMA is very similar to the security notion we have for MAC.

7.7 Towards Post-Quantum Cryptography

All public-key algorithms we have seen so far are either based on the factoring problem or the discrete logarithm problem. Both problems can easily be solved with quantum computers [78]. When these devices will become available (although it is impossible to predict when and if this will be the case), we will not have any security when using these algorithms. For that, we need alternate schemes based on different problems.

So far, the most promising directions seem to rely either on problems from coding theory (such as the McEliece cryptosystem [58] or the TCHo cryptosystem [34]), or based on lattices (such as NTRU [44] or Regev [67]).

Lattices are discrete subgroup of the vector space $\mathbf{R}^m$, or more simply, the set of the linear combinations with integral coefficients of vectors from a basis of the lattice.

$$\mathcal{L}(\vec{a}_1, \ldots, \vec{a}_n) = \left\{ \sum_{i=1}^{n} s_i \vec{a}_i \ ; \ s_1, \ldots, s_n \in \mathbf{Z} \right\}$$

In this structure, given the basis $\vec{a}_1, \ldots, \vec{a}_n$, it is hard to find short non-zero vectors $\vec{x} \in \mathcal{L}(\vec{a}_1, \ldots, \vec{a}_n)$. Given the basis and some vector $\vec{b} \in \mathbf{R}^m$, it is also hard to find $\vec{x} \in \mathcal{L}(\vec{a}_1, \ldots, \vec{a}_n)$ making $\|\vec{b} - \vec{x}\|$

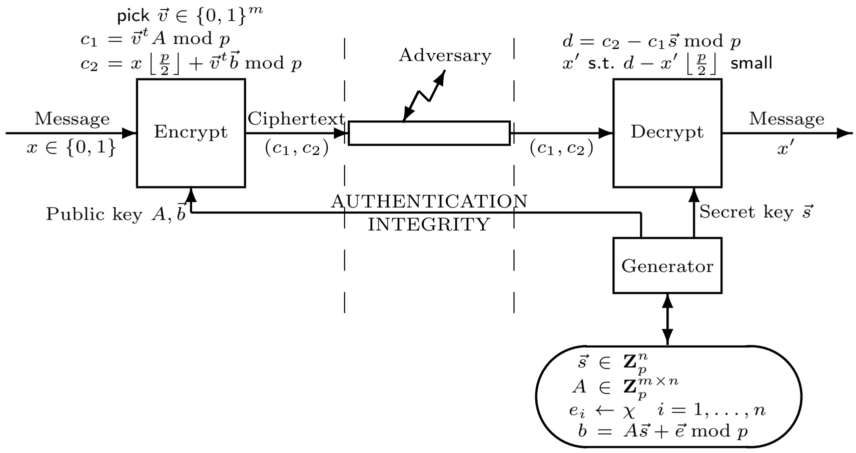

Figure 7.12: The Regev Public-Key Cryptosystem

small. Many cryptographic algorithms are based on lattices. We can even construct some *fully homomorphic encryption*, i.e. some encryption Enc such that we have public functions f_+ and $f_\times$ such that for all x and y,

$$\mathsf{Dec}\left(f_+(\mathsf{Enc}(x), \mathsf{Enc}(y))\right) = x + y$$

and

$$\mathsf{Dec}\left(f_\times(\mathsf{Enc}(x), \mathsf{Enc}(y))\right) = xy$$

or at least with high probability. Based on f_+ and $f_\times$, we can compute any polynomial function on encrypted data, so consider secure outsourced computations. One problem with lattice-based cryptography is that the size of public keys is typically large. Nevertheless, it is likely to become standard in the near future.

As an example, we describe the cryptosystem of Regev [67] on Fig. 7.12. First, we have some common parameters which consist of some prime p, some real $\varepsilon > 0$, some integers m and n such that $n^2 \leq p \leq 2n^2$ and $m = (1+\varepsilon)(n+1)\log_2 p$, and some real number $\alpha = \frac{1}{\sqrt{n}\log_2^2 n}$. If X is a random variable with normal distribution $\mathcal{N}(0, \alpha p)$ with expected value 0 and standard deviation αp, we denote by χ the distribution of the random variable $\lfloor X \rceil$ obtained by rounding X to the nearest integer. (So, the order of magnitude of a random variable following χ is $4\sqrt{p}/\log_2^2 p$.) A secret key is just a vector $\vec{s} \in \mathbf{Z}_p^n$. To construct a public key, we first pick a random $m \times n$ matrix A with coefficients in $\mathbf{Z}_p$. Then, we pick a vector $\vec{e}$ of length m with random independent coefficients following χ. (So, $\vec{e}$ has a short norm.) We set $\vec{b} = A\vec{s} + \vec{e} \bmod p$. (So, $\vec{b}$ is close to $A\vec{s}$ and $A\vec{s}$ leaks $\vec{s}$ because m is larger than n. Clearly, we really need the search for a close vector to be hard.) The public key is the pair $(A, \vec{b})$. To encrypt a bit x, we compute the pair (c_1, c_2) defined by first picking a random row $(v_1, \ldots, v_m)$ of bits, then by $c_1 = \sum_{i=1}^m v_i A_i \bmod p$ (where A_i is the ith row of A) and $c_2 = x\left\lfloor \frac{p}{2} \right\rfloor + \sum_{i=1}^m v_i b_i \bmod p$. To decrypt (c_1, c_2), we compute $d = c_2 - c_1 \vec{s} \bmod p$, which is normally equal to $x\left\lfloor \frac{p}{2} \right\rfloor - \sum_{i=1}^m v_i e_i \bmod p$, so close to $x\left\lfloor \frac{p}{2} \right\rfloor$. Then, we check if it is closer to 0 than $\left\lfloor \frac{p}{2} \right\rfloor$ to deduce x.

The Regev scheme inspired many others which are essentially some variants of the following construction.

We define the following algebra. We consider six *additive Abelian groups* S_{sk}, S_A, S_B, S_t, S_U, and S_V and four *bilinear mappings* which are all denoted with $\times$: $S_A \times S_{\mathsf{sk}} \to S_B$, $S_U \times S_{\mathsf{sk}} \to S_V$, $S_t \times S_A \to S_U$, and $S_t \times S_B \to S_V$. We assume associativity in the sense that

$$(t \times A) \times \mathsf{sk} = t \times (A \times \mathsf{sk})$$

for all $t \in S_t$, $A \in S_A$, and $\mathsf{sk} \in S_{\mathsf{sk}}$. We also assume that there is a norm $\|\cdot\|$ on S_{sk}, S_B, S_t, S_U, and S_V. (It is symmetric, positive, and satisfies the triangular inequality.) We assume we can

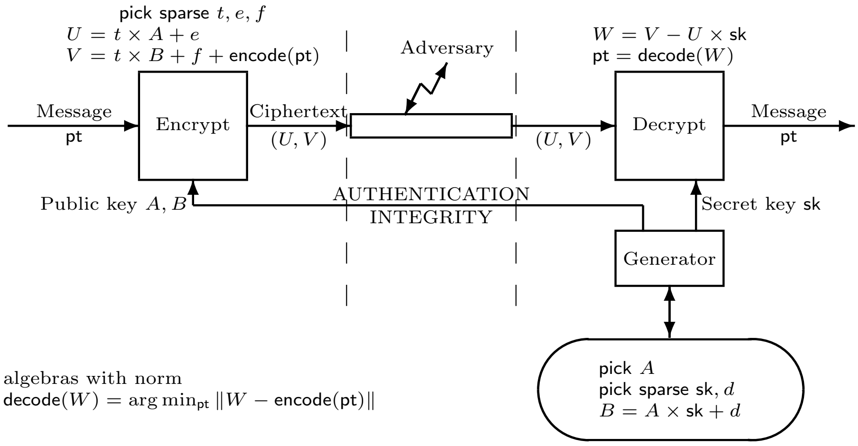

Figure 7.13: A Meta Postquantum Cryptosystem

upper bound $\|x \times y\|$ in terms of $\|x\|$ and $\|y\|$ for the four bilinear functions. Finally, we assume two functions $\mathsf{encode} : \mathcal{M} \to S_V$ and $\mathsf{decode} : S_V \to \mathcal{M}$ such that encode is injective. The image set $C = \mathsf{encode}(\mathcal{M})$ is called a *code*. Decoding finds *one* closest codeword in the following sense:

$$\forall W \in S_V \quad \mathsf{decode}(W) = \arg\min_{\mathsf{pt}} \|W - \mathsf{encode}(\mathsf{pt})\|$$

Informally, elements with a small norm are called *sparse*. In what follows, we use small letters to designate sparse elements. We define a PKC as follows in which the choice of the algebra, norm, encoding/decoding, and the probability distributions are left free:

Algorithm gen(coinA):
pick a random $A \in S_A$ and random *sparse* $\mathsf{sk} \in S_{\mathsf{sk}}$ and $d \in S_B$ by using coinA
$B \leftarrow A \times \mathsf{sk} + d$
$\mathsf{pk} \leftarrow (A, B)$
return $(\mathsf{sk}, \mathsf{pk})$

Algorithm enc(pk, pt; coinB):
parse $\mathsf{pk} = (A, B)$
pick random *sparse* $t \in S_t$, $e \in S_U$, and $f \in S_V$ by using coinB
$U \leftarrow t \times A + e$
$V \leftarrow t \times B + f + \mathsf{encode}(\mathsf{pt})$
return $\mathsf{ct} = (U, V)$

Algorithm dec(sk, ct):
parse $\mathsf{ct} = (U, V)$
$W \leftarrow V - U \times \mathsf{sk}$
$\mathsf{pt}' \leftarrow \mathsf{decode}(W)$.
return pt'

Thanks to bilinearity and associativity, we have $W = \delta + \mathsf{encode}(\mathsf{pt})$ with

$$\delta = t \times d + f - e \times \mathsf{sk} \tag{7.1}$$

This value δ will be called the *noise*. By controlling (with their respective probability distribution) the size of t, d, f, e, sk, we can make sure that the noise δ is sparse. Hence, $\mathsf{decode}(W) = \mathsf{pt}$. The scheme is depicted on Fig. 7.13.

We mention two examples of real cryptosystems based on this construction.

- *FrodoPKE-640*: we set $q = 2^{15}$, $\bar{m} = \bar{n} = 8$, $n = 640$, and $\ell = 2$. Then, the algebra is defined by

$$S_{\mathsf{sk}} = S_B = \mathbf{Z}_q^{n \times \bar{n}} \quad S_A = \mathbf{Z}_q^{n^2} \quad S_t = S_U = \mathbf{Z}_q^{\bar{m} \times n} \quad S_V = \mathbf{Z}_q^{\bar{m} \times \bar{n}}$$

So, elements are matrices of various sizes, with coefficients in $\mathbf{Z}_q$. The bilinear functions are matrix multiplications. The norm is defined by

$$\|X\| = \max_{i,j} \left| \left(\left(X_{i,j} + \frac{q}{2} \right) \bmod q \right) - \frac{q}{2} \right|$$

and encoding works as follows:

$$(\mathsf{encode}(\mathsf{pt}))_{i,j} = q2^{-\ell} \sum_{k=1}^{\ell} 2^{k-1} \mathsf{pt}_{\ell((i-1)\bar{n}+(j-1))+k}$$

This means that each matrix element encodes ℓ bits of the plaintext.

- *NewHope512CPA-PKE*: We set $q = 12\,289$ and $n = 512$. Then, the algebra is defined by

  $$S_{\mathsf{sk}} = S_A = S_B = S_t = S_U = S_V = \mathbf{Z}_q[z]/(z^n + 1)$$

  So, elements are polynomials in z modulo $z^n + 1$ and modulo q. Hence, they are represented as polynomials with degree bounded by $n - 1$ and coefficients in $\mathbf{Z}_q$. The bilinear functions are defined by the multiplication in this structure, i.e. the multiplication of polynomials modulo $z^n + 1$ and modulo q. The norm is defined by

  $$\left\| \sum_{i=0}^{n-1} X_i z^i \right\| = \max_i \left| \left(\left(X_i + \frac{q}{2} \right) \bmod q \right) - \frac{q}{2} \right|$$

  and encoding works as follows:

  $$\mathsf{encode}(\mathsf{pt}) = \frac{q}{2} \sum_{i=1}^{n} (z^{i-1} + z^{i+255}) \mathsf{pt}_i$$

  This means that each bit of the plaintext appears as the most significant bit of two elements.

In 2022, NIST selected for the post-quantum cryptography standard one encryption algorithm and three digital signature schemes. The encryption algorithm is CRYSTALS-KYBER. It is based on lattices and follows the above meta-structure. Like for NewHopes, it uses $\mathbf{Z}_q[z]/(z^n + 1)$ but for $q = 3329$ and $n = 256$. In addition to this, it uses heavily optimized algorithms. For instance, instead of representing polynomials in $\mathbf{Z}_q[z]/(z^n+1)$ by their sequence of coefficients, it represents them through the so-called *NTT transform* which enables faster multiplication.

The three NIST selected signature schemes are CRYSTALS-DILITHIUM, FALCON, and SPHINCS+. The first two are based on lattices. The last one is so called *hash-based*.

The Fujisaki-Okamoto transform. Most of the cryptosystems start with a weakly secure (typically: INDCPA-secure) scheme which is transformed by a strongly secure one (typically: INDCCA) using standard techniques. These techniques are essentially based on the one proposed by Fujisaki and Okamoto in 1999 [39, 40]. Essentially, they start from a cryptosystem $(\mathsf{Gen}_0, \mathsf{Enc}_0, \mathsf{Dec}_0)$ which is secure against decryption under CPA and also γ-spread. This latter notion means that there is no ciphertext value which is taken too often. More precisely

$$\forall \mathsf{pk}, \mathsf{pt}, \mathsf{ct} \qquad \Pr[\mathsf{Enc}_{\mathsf{pk}}(\mathsf{pt}) = \mathsf{ct}] \leq 2^{-\gamma}$$

The construction also uses a one-time secure cipher (which we take as one-time pad below) and two random oracles G and H. The new cryptosystem is defined with $\mathsf{Gen} = \mathsf{Gen}_0$ as follows:

$\mathsf{Enc}_{\mathsf{pk}}(\mathsf{pt})$:
1: pick σ
2: $\mathsf{ct}_2 \leftarrow \mathsf{pt} \oplus G(\sigma)$
3: $\mathsf{ct}_1 \leftarrow \mathsf{Enc}_{0,\mathsf{pk}}(\sigma; H(\sigma, \mathsf{ct}_2))$
4: **return** $(\mathsf{ct}_1, \mathsf{ct}_2)$

$\mathsf{Dec}_{\mathsf{sk}}(\mathsf{ct}_1, \mathsf{ct}_2)$:
1: $\sigma \leftarrow \mathsf{Dec}_{0,\mathsf{sk}}(\mathsf{ct}_1)$
2: **if** $\sigma = \perp$ **then return** $\perp$
3: **if** $\mathsf{ct}_1 \neq \mathsf{Enc}_{0,\mathsf{pk}}(\sigma; H(\sigma, \mathsf{ct}_2))$ **then return** $\perp$
4: $\mathsf{pt} \leftarrow \mathsf{ct}_2 \oplus G(\sigma)$
5: **return** pt

Hash-based signatures. Unlike encryption, we can build a signature scheme from a one-way function only.

One key idea dates from the Lamport signature scheme which can be used only once, to sign an n-bit message: the secret key is a tuple of $2n$ random values $(\mathsf{sk}_{i,b})_{i,b}$ for $b \in \{0,1\}$ and $i = 1, \ldots, n$. The public key is the tuple of all $\mathsf{pk}_{i,b} = \mathsf{OW}(\mathsf{sk}_{i,b})$, following a one-way function OW. To sign an n-bit message $m_1 \cdots m_n$, the signer reveals $\sigma_i = \mathsf{sk}_{i,m_i}$ for $i = 1, \ldots, n$. To verify the signature, the verifier checks that $\mathsf{pk}_{i,m_i} = \mathsf{OW}(\sigma_i)$ for every i.

Many improvements are possible. A substantial one was made to enable a signature scheme which could be used a few times instead of only once. The secret key is a tuple of kt random values $(\mathsf{sk}_{i,j})_{i,j}$ for $i = 1, \ldots, k$ and $j = 1, \ldots, t$. The public key is the tuple of all $\mathsf{pk}_{i,j} = \mathsf{OW}(\mathsf{sk}_{i,j})$. To sign a message m, we first hash it and parse the result $H(m)$ as a sequence $j_1, \ldots, j_k$ of indices $j_i \in \{1, \ldots, t\}$. In other words, the hash is a sequance of k indices between 1 and t. The signature is the tuple of all $\sigma_i = \mathsf{sk}_{i,j_i}$ for $i = 1, \ldots, k$. To verify the signature, the verifier (after hashing the message) checks that $\mathsf{pk}_{i,j_i} = \mathsf{OW}(\sigma_i)$ for every i.

In both constructions, the public key can be compressed into a single hash by using a *Merkle tree*. The idea is that every value to authenticate is put on a leave of a binary tree. In this tree, the values of the children of a node are hashed together in order to define the value of the parent. Hence, by propagating the hashes, we obtain the value of the root of the tree. This value is the compression of all others. In order to authenticate a single value, we provide the valus of the sibling of every ancestor so that the value of the root can be computed again. Provided that the hash function is collision-resistant, this is a secure way to compress.

The SPHINCS+ signature scheme is based on all these techniques. It uses a Merkle tree to authenticate many public keys. Each key is a compressed public key of the few-time signature scheme. To sign a message, it is first hashed together with a random value (to be provided in the signature) in order to define a digest and the index of a compressed key to use. Then, this key is used to sign the digest as above.

7.8 Other Primitives

There is a variant of the public-key cryptosystem primitive called *key encapsulation mechanism (KEM)* [30]. In this primitive, there are again three algorithms: a key generation, an encryption KemEnc, and a decryption algorithm KemDec. Key generation and decryption are similar as in public-key cryptosystems. Encryption however is not used to encrypt a message. Instead, running the KEM encryption algorithm KemEnc with a public key pk produces a plaintext K and a ciphertext C. I.e., there is no control on which plaintext K is encrypted. The produced plaintext K is just random.

The plaintext K is meant to be used as a symmetric key in *hybrid encryption* (see Fig. 7.14). Indeed, KEM is to be used with a *data encapsulation mechanism (DEM)* which can encrypt or decrypt a message with a symmetric key. So, with KEM and DEM together, we DEM-encrypt a message with the key produced by the KEM-encryption and we concatenate the two ciphertexts (the one from KEM and the one from DEM). To decrypt, we first apply the KEM-decryption to recover the symmetric key then run the DEM-decryption to recover the message.

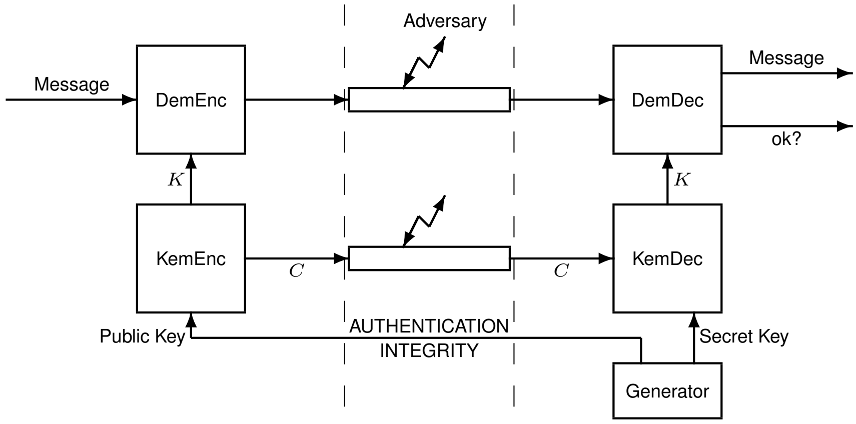

Figure 7.14: Hybrid Encryption using KEM and DEM

Another useful primitive which can be met in cryptographic protocols is the *commitment scheme*. We have already seen some construction based on hash functions. We can also propose some based on the discrete logarithm problem.

In the *Pedersen commitment* [62], we use a subgroup of $\mathbf{Z}_p^*$ of prime order q, proposed with two generators g and h. To commit on a number X, we pick $r \in \mathbf{Z}_q$ and disclose $c = g^X h^r \bmod p$. Opening the commitment consists of revealing X and r and checking the correctness of c.

This commitment scheme is unconditionally hiding: the distribution of c is statistically independent from X. The commitment is also computationally binding: being able to open c on two different values X and X' is equivalent to computing the discrete logarithm of h in basis g. Indeed, if $c = g^X h^r \bmod p$ and $c = g^{X'} h^{r'} \bmod p$, $a = \frac{X'-X}{r-r'} \bmod q$ is such that $h = g^a \bmod p$.

Chapter 8

Trust Establishment

This chapter shows how to build trusted information infrastructure using cryptographic primitives. It focuses on access control, secure password-based key setup, secure communication, and key infrastructures.

8.1 Access Control

Password access control. Access control can be done by sending an identifier ID and a password w. To avoid leakage from the database, we can avoid storing the password by saving only a hash of it. To secure against multi-target attacks and time-memory tradeoffs, we can add a *salt* which has to be stored as well. So, the database contains some $(\mathsf{ID}, \mathsf{salt}, H(\mathsf{ID}\|\mathsf{salt}\|w))$ triplets (see Fig. 8.1).

The advantage is that the server does not keep w and that the client does not need to compute anything (which is nice for a human client). The drawback is that the channel to transmit w must be secure.

Challenge/response access control. Another popular technique consists of sending a challenge c to the client to which he must answer by some $r = f_K(c)$, where K is his key and f is a *pseudorandom function (PRF).* This requires that the server keeps K in a database. The client has to do some computation. This is rather done by a device than a human being. For instance, the client can be a special hardware (e.g., a SIM card, for GSM telephony, see Section A.3). It can work with high-entropy shared secret K.

The advantage is that it resists to passive adversaries. The drawback is that it does not resist to relay attacks. In addition to this, the client must do some cryptographic operation. This cannot be done by a human client.

One-time password (OTP). In between password access control and challenge/response protocols, we have the *one-time password* protocol. Essentially, we have a long list of passwords and each password can only be used only once. There may also be a chronological order to use these passwords. Typically, the sequence is generated from a secret seed backward: from the last-to-be-used password to the first one. Each password is the image of the next password by a one-way function. So, the server only stores the last used password and checks that the new one hashes onto the stored one.

This resists to passive adversaries, not to relay attacks, it requires no computation on the client side, but managing a long list of password may not always be well accepted by human users.

Strong Authentication. We call *strong authentication* the techniques using several factors. We can use factors based on *what we know* (e.g., a password), on *what we process* (e.g., a secure

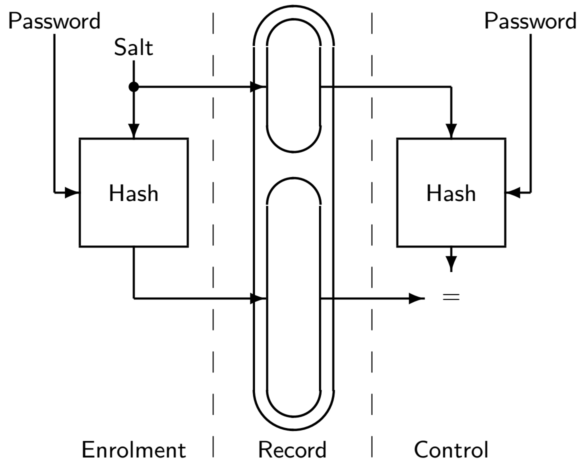

Figure 8.1: Password Access Control with Salt

token), or on *what we are* (e.g., with biometric recognition). For example, a smart card doing some authentication and that has to be unlocked by a PIN code is a strong authentication.

8.2 Password-Based Cryptography

Since passwords have low entropy, we must live with *online attacks* in which an adversary impersonates a honest user by guessing the password. What we want to avoid is that the protocol leaks some information which could be used as a stop test predicate in an *offline* exhaustive search.

In access control, if we use a challenge/response protocol like above, it leaks some information making offline exhaustive search easy. If we run the Diffie-Hellman protocol where the exchanged public keys are authenticated using a MAC with the password as a key, this also leaks some information leading to an offline attack.

There exists protocols not leaking information still doing the authentication. Interestingly, we can combine the password-based authentication with key agreement to set up an ephemeral symmetric key. This cryptographic primitive is called a *password-based authenticated key exchange* (PAKE).

For instance, in the EKE protocol by Bellovin and Merritt [15], Alice generates an ephemeral secret/public key pair, encrypts her public key with the password and sends it to Bob. Bob can decrypt it with the password, then use it to encrypt the session key K. The encrypted key can be re-encrypted using the password and sent to Alice. Then, she can recover the encrypted key and decrypt it with her ephemeral secret key to obtain K (see Fig. 8.2). This is an *authenticated key agreement protocol*. Note that the choice for the public-key encryption and symmetric encryption may be tricky as many instances are weak.

It is indeed very delicate to construct a secure protocol. With the EKE paradigm, we could easily imagine to use RSA with a constant exponent e and encrypt the modulus N. Let C be the encrypted modulus. As a result, and guess w for the password such that $\mathsf{Dec}_w(C)$ has small prime factors (e.g. it is even) would corresponds to an invalid RSA modulus. Hence, those guesses for w could be ruled out and we could keep a small partition of plausible w value. This attack is called a *partition attack*. Its principle is to rule out many possible w. By iterating the attack, we can then isolate the unique possible value of w.

There exists many password-based authenticated key agreement protocols. One popular one (in standards) is the Secure Remote Password (SRP) protocol [92]. Probably, the simplest and most elegant one is SPAKE2 [10]. (See Fig. 8.3.)

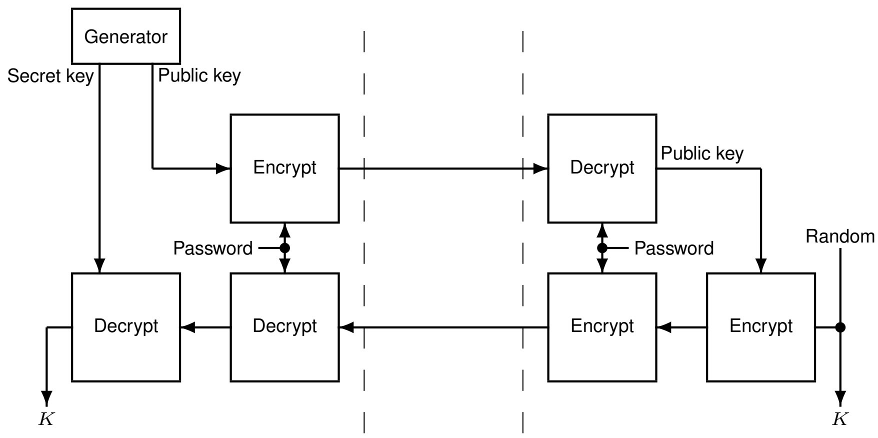

Figure 8.2: The EKE Protocol

$$
\begin{array}{rcl}
\textbf{Alice} & & \textbf{Bob} \\
\textbf{password}\text{: } w & & \textbf{password}\text{: } w \\
& & \\
\text{pick } x \in \mathbf{Z}_q^*,\ X \leftarrow g^x & & \text{pick } y \in \mathbf{Z}_q^*,\ Y \leftarrow g^y \\
X^* \leftarrow X g_2^w & \xrightarrow{X^*} & Y^* \leftarrow Y g_3^w \\
K_A \leftarrow (Y^*/g_3^w)^x & \xleftarrow{Y^*} & K_B \leftarrow (X^*/g_2^w)^y \\
K'_A \leftarrow \mathsf{KDF}(A, B, X^*, Y^*, w, K_A) & & K'_B \leftarrow \mathsf{KDF}(A, B, X^*, Y^*, w, K_B) \\
& & \\
\textbf{output}\text{: } K'_A & & \textbf{output}\text{: } K'_B
\end{array}
$$

(public parameters: q prime, g, g_2, g_3 generators of the same order-q group)

Figure 8.3: The SPAKE2 Password-Based Key Exchange Protocol

8.3 From Secure Channel to Secure Communications

The main security properties that we must obtain for secure communication are:

- *confidentiality*: only the legitimate receiver can retrieve the message;
- *authentication*: only the legitimate sender can create a new message;
- *integrity*: the received message must be equal to the sent one.

Confidentiality is enforced by symmetric encryption. We could enforce integrity using a hash function (but the digest must be sent with integrity protection), but it is not necessary since integrity and authentication are enforced at the same time by a MAC. We can however find some unorthodox authentication means which do not protect integrity. For instance, people sometimes say that GSM, Bluetooth and WiFi authenticate by encryption: no adversary can create a new message which can make sense after decryption. Indeed, the idea is to use one-time-pad on the message X concatenated with a (linear) CRC function. An adversary could however modify an existing message so that it will not be decrypted like the original one. So, integrity is not protected.[1] Good authentication means (such as the ones based on a MAC) should be preferred as they protect integrity at the same time. Finally, we can combine symmetric encryption and MAC (or use some authenticated encryption methods) to obtain the three properties at the same time.

There are other properties that we need for secure communication, depending on applications:

- *freshness*: when a message is received, the receiver would like to be ensured that it is a new (fresh) message and not a replayed one;
- *liveliness*: when a message is sent, the sender would like to be ensured that it will be delivered, eventually;
- *timeliness*: in addition to liveliness, the sender would like the delivery time to be bounded;
- *deniability*: no evidence of sending a message leaks;
- *non-repudiation*: the sender of a message cannot prove he did not send his message (this is somehow opposite to deniability);
- *forward secrecy*: secrecy remains even if some long-term keys leak in the future;
- *postcompromise security*: security can heal after some secret keys leak.

These properties must be satisfied at the packet level. When it comes to order the packets in a communication session, we need further properties:

- *key establishment*: we need a way to set up the symmetric key which will be used during the session;
- *session integrity*: we must guaranty that the sequence of packets is the same on both sides, to avoid attacks based on packet swaps or packet drops;
- *privacy*: sometimes, we want to hide the identity of the sender and the receiver, or just the fact that some sessions are made by the same person, or even the total duration or volume of communication. There are many notions of privacy.

When it comes to address session integrity, there is a property which is easy to obtain: *sequentiality*. It makes sure that whenever a participant has received a message X_t in a sequence of messages $X_1, \ldots, X_t$, then his counterpart must have seen the exact same sequence of messages, at least in the past (i.e., maybe he has sent another message which has not been delivered yet). Sequentiality

[1] This is also discussed in Section A.1 about WiFi.

can be obtained by using packet counters and by authenticating the counters. For instance, in the previous versions of TLS, a packet X was set as

$$Y = \mathsf{Enc}\left(X \| \mathsf{MAC}(\mathsf{seq} \| X)\right)$$

where seq is a synchronized packet counter. In TLS 1.3, we use

$$Y = \mathsf{AE.Enc}(\mathsf{seq}, X)$$

with authenticated encryption AE, where seq is the additional data to be authenticated.[2] What is missing to obtain full session integrity is the notion of *termination fairness*: to make sure that the last message of the communication is the same.

Fair termination can be required in a contract signing protocol: both parties want to terminate the protocol by agreeing on either the contract is valid or the transaction failed. We do not want that one participant thinks there is a contract and the other thinks the negotiation aborted. For this, they essentially need to terminate with a bit which must be the same on both ends. This can be done with a synchronization protocol such as the *keep-in-touch protocol* [13]: both participants have a bit. For a participant, if the bit is 0, he does nothing. Otherwise he goes through the keep-in-touch protocol. If it completes, the final bit is set to 1. In the case of any time-out in the protocol, a participant changes his bit to 0 and aborts the protocol. In the protocol, Alice picks a random number N and sends it encrypted. Then, both participants exchange a total number of messages equal to N. After the Nth message is sent, the sending participant waits for a delay before completing the protocol. His counterpart can complete upon reception of the Nth message. It was proven that if the channel is already secure (but for termination fairness) and we want that both participants end on the same bit except with a probability bounded by some p, then we need an average number of exchanged messages to be $\Omega(\frac{1}{p})$. This is for any protocol and the keep-in-touch protocol achieves this bound. So, it is pretty expensive to achieve termination fairness in general. In most of secure channel implementations, this is not done.

8.4 Setup of Secure Channels

To setup a symmetric key for a secure channel, we need a (less) secure channel. Indeed, with public-key cryptography, we only need an authenticated-integer channel to transmit a public key. Then, we can do key transfer or key exchange using a public-key cryptosystem or a key agreement protocol. If we don't have this authenticated-integer channel, we can still use the insecure channel but it will not protect against active adversaries playing the man-in-the-middle. We will still be protected against passive adversaries who only see the exchanged messages without interfering.

In practice, this authenticated-integer channel is either a real secure channel (e.g. a setup cable between two devices, or some short range wireless technology such as NFC) or relying on a third participant such as a human user (like in the Bluetooth pairing which is presented in Section A.7), a secure token (e.g., a SIM card), a key server or a certificate authority.

Finally, we can say that, except for the beginning and the end of a secure conversation (where we must setup a key and have fair termination, respectively), all other properties are pretty easy to achieve using good cryptography.

8.5 Setup by Narrowband Secure Channel

Secure communication (by means of symmetric cryptography) requires to set up a symmetric key through a fully secure channel. We can relax the confidentiality requirement on this channel by using public-key cryptography, but this still requires to set up a public key through a channel protecting authentication and integrity.

[2] See Section A.5.

Public keys may be large (e.g. too large to be spelled by human beings). We could rely on a fully secure narrowband channel and use a password-based authenticated key agreement protocol. The secure narrowband channel would be used to set up a password. We have seen in a previous chapter how these primitives can set up a symmetric key using a securely set up password, even a password with low entropy.

The final step to complete the picture is to relax the confidentiality requirement on the narrowband secure channel. Using a narrowband channel protecting only authentication and integrity, we can transmit a *short authenticated string (SAS)*.

The Bluetooth example (which is presented in Section A.7) illustrates a technique which is quite interesting: to set up a secure communication channel by using a short string which was authenticated through an alternate channel (here: user monitoring). This short string actually a SAS. This type of protocol is used to set up personal area networks (e.g., with Bluetooth SSP in the version 2.1 or in the 802.11 standard in the version 3). It is also used to manually authenticate public keys. For instance, this is used in voice over IP protocol (e.g. ZRTP).

In a *message authentication protocol*, one sender wants to authenticate a message of arbitrary length by using a SAS. An example is when people print a SAS on their visit card which could be used to authenticate their public key, which can be retrieved online (over an insecure channel). Typically, we just authenticate the digest of the public key. Instead, by authenticating the hash of the commitment on the message (with public opening), we can have a secure protocol without relying on the collision resistance on the hash function. I.e., we could hash onto 80 bits only. To shrink the SAS further, we need interaction. One message authentication protocol was proposed by Vaudenay in 2005 [82] and proven secure, even with a SAS of 20 bits. Since then, other protocols have appeared.

8.6 Setup by a Trusted Third Party

Setting up a key using a third party can have several forms. First, we have seen human users playing the role of a third party (with a password or a SAS), which could make sense in some applications. We can have a pervasive third party in the form of a secure token such as a smart card, a secureID device, or a trusted computing platform (TPM). We can also use remote services. With Kerberos [50], we have a key server to help two participants to communicate. With public-key infrastructures (PKI), we can rely on a certificate authority.

In Kerberos, every participant (server or client) shares a symmetric key with the authority. When Alice wants to talk to Bob, she sends a request to the authority who will select a symmetric key K for them. Then, it will encrypt for Alice a "ticket" and K. The ticket contains K and is encrypted for Bob. In addition to this, it also contains the identity of Alice and a validity period for using K.

In PKIs, the certificate authority has a public key which is assumed to be securely distributed. A server setting up his key must securely deposit his public key to the authority. The authority will in return sign a certificate assessing that the key was well deposited by this server. Then, a client receiving this certificate can check the signature and extract the public key of the server (see Fig. 8.4). This is typically used to establish a semi-authenticated channel: the client authenticates the server, then they set up a secure channel.

In practice, things do not work so smoothly. Indeed, there exist many certificate authorities and they are all equally recognized by browsers. Some servers may lose their keys, meaning that the (valid) certificate must be revoked. There are essentially two approaches for that: having regular certificate revocation lists (CRL) issued by the authority or using the online certificate status protocol (OCSP). Another problem is that some authorities may be corrupted.

There exist alternatives to the PKI model. For instance, in the identity-based encryption model [24], the authority issues common parameters (see Fig. 8.5). To encrypt to Bob, Alice needs these common parameters, Bob's identity and the current time period in the system. To decrypt, Bob receives a new secret key from the authority every time period. This model has the problem that the authority owns the secret key of every user, but we can mix up this model with

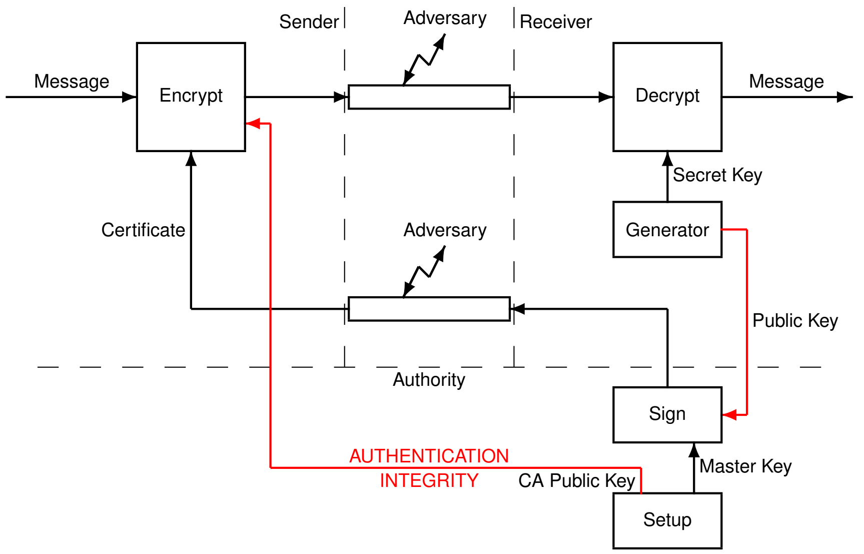

Figure 8.4: The PKI Model

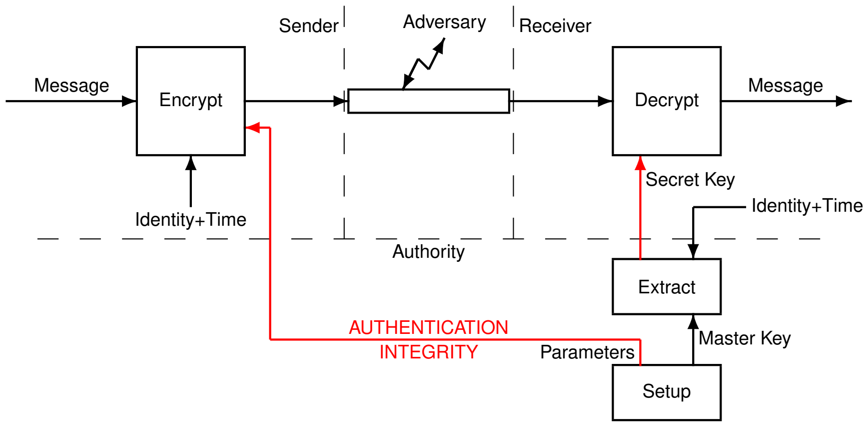

Figure 8.5: The Identity-Based Cryptography Model

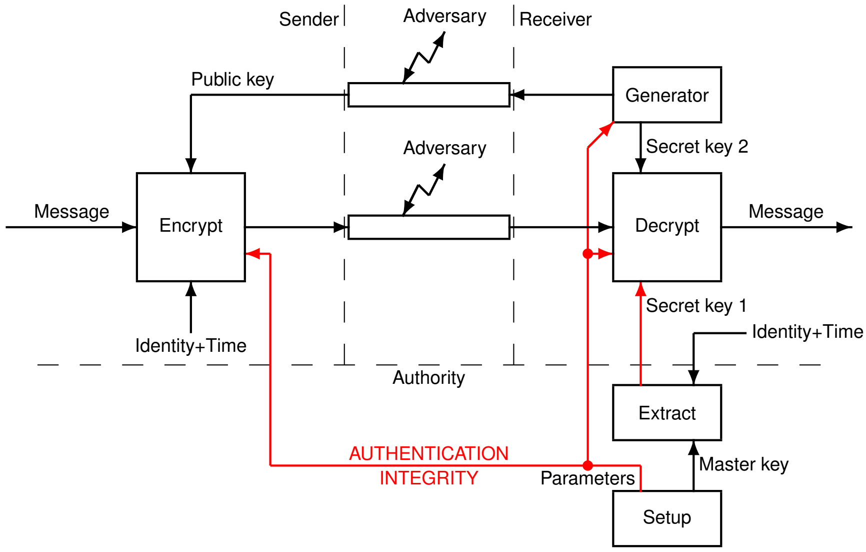

Figure 8.6: The Certificateless Cryptography Model

others. In the certificateless encryption model [12], Alice needs the common parameters, Bob's identity and the time period, and Bob's public key (which requires no certificate in this model). To decrypt, Bob needs the secret given by the authority and his own secret key (see Fig. 8.6). So, this model resists to two possible attacks: adversaries modifying the public key and honest-but-curious authorities (who do not modify the public key).

As an example of a concrete construction, we mention the Boneh-Franklin [24] identity-based encryption scheme. It is the very first efficient scheme of this type. It is based on a pairing e, in a group of prime order q generated by some point P. The scheme is depicted on Fig. 8.7. It is set up by picking a master secret $s \in \mathbf{Z}_q^*$ then computing the main public parameter $K\mathsf{pub} = sP$. Encryption of a message m for an identity ID is made by first extracting the public key $Q_{\mathsf{ID}} = H_1(\mathsf{ID})$ (meaning that a public function H_1 generates a group element from a string ID), picking a random $r \in \mathbf{Z}_q^*$. The encryption computes $u = rP$ and $v = m \oplus H_2(e(Q_{\mathsf{ID}}, K_{\mathsf{pub}})^r)$. The ciphertext is (u, v). To decrypt, one needs the secret key $d_{\mathsf{ID}} = sQ_{\mathsf{ID}}$. (As we can see, the authority can compute the secret of any user.) Essentially, we use that

$$e(Q_{\mathsf{ID}}, K_{\mathsf{pub}})^r = e(Q_{\mathsf{ID}}, sP)^r = e(sQ_{\mathsf{ID}}, rP) = e(d_{\mathsf{ID}}, u)$$

to decrypt $m = v \oplus H_2(e(d_{\mathsf{ID}}, u))$.

8.7 Trust Management and Cryptography

The PKI model is very fragile as it requires to rely on a very long trust chain: the certificate authority, the browser editor, the hardware manufacturer, the retailer, the operating system in its real environment, and the human user who is not always careful.

The consequence is that there are phishing attacks on the network: some hackers make fake servers imitating the real ones, with either no secure connection (which is not always detected) or a secure one with an untrusted certificate (which may be accepted by the human user anyway), or with a revoked certificate or a certificate from a corrupted authority.

The PKI model is mainly the one used by TLS.[3] The SSH protocol relies more on the fact that the public key does not change over time: the first connection may be problematic, but then the public key is locally stored and the client checks that it does not change. With the PGP software,

[3] See Section A.5.

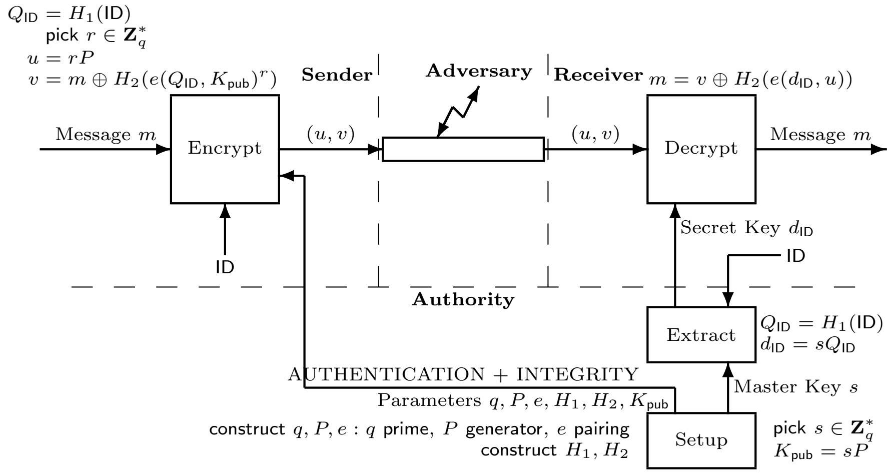

Figure 8.7: Boneh-Franklin Identity-Based Encryption

public keys are cross-signed by a community of users. So, new keys may be checked based on reputation, in addition to comparison with a local key ring like with SSH.

There are other causes of security problems. For instance, the lack of cryptographic diversity (everybody is using the same algorithms) makes the algorithms highly exposed. If one is broken, security may collapse. Some cryptographic protocols badly interact and their composition may be insecure even though they may be secure alone. Adversaries are making constant progresses, with their equipment, knowledge about algorithms, etc. Finally, many cryptographic algorithms are published with proofs which are incorrect, or models which are not appropriate, and it takes a long time before we realize it.

On top of these, the quantum threat is there. We know for sure that most of cryptographic schemes that we use today will be broken once quantum computers will be built. This means that the data which is sent today will eventually be decrypted. There is an urgent need for good cryptography which resists quantum attacks: post-quantum cryptography.

Appendix A

Case Studies

A.1 WiFi: WEP/WPA/WPA2

WEP. Before 2003, WiFi was secured by WEP, and sometimes by SKA as well. Both use an encryption function in which the ciphertext consists of a nonce IV and the XOR between the plaintext and the keystream generated by RC4, in which K is the concatenation of IV and the secret key. This key is pre-shared. The nonce is pretty small: 3 bytes. So, it eventually repeats. Also, the access point can manage up to 4 keys. So, the keys are eventually shared by many devices.

The SKA (Shared Key Authentication) algorithm was used to authenticate the host to the access point: the access point was sending a challenge and the host had to respond with the encryption of this challenge. Clearly, this leaks to an observer some IV and the keystream generated by this IV. So, it allows to decrypt communications using the same IV! Due to this terrible weakness, SKA stopped to be used at all. So, we had no peer authentication.

The WEP (Wired-Equivalent Privacy) protocol consists of encrypting communication.

There is an integrity protection based on a $\mathsf{CRC32}$ algorithm: the message x is concatenated with a $\mathsf{CRC32}(x)$ function. But since $\mathsf{CRC32}$ is completely linear, the protection is void [26]: the encryption of x would be

$$y = (x\|\mathsf{CRC32}(x)) \oplus \mathsf{keystream}$$

but the encryption of $x \oplus \delta$ would be

$$y \oplus (\delta\|\mathsf{CRC32}(\delta)) = ((x \oplus \delta)\|\mathsf{CRC32}(x \oplus \delta)) \oplus \mathsf{keystream}$$

So, an adversary willing to substitute $x \oplus \delta$ to x would just substitute $y \oplus \delta\|\mathsf{CRC32}(\delta)$ to y, which is doable without knowing x or the key: we only need the ciphertext and δ.

There are also obvious weaknesses due to repetition of IV. In addition to this, there are some key recovery known-plaintext attacks which are really impressive (it only requires 20 000 packets [75]). So, there is almost no security.

WPA. In 2003, a new system called WPA-TKIP (Temporal Key Integrity Protocol) was introduced to regularly modify the value of K in the RC4 encryption. In addition to this, alternatives to the pre-shared key system were introduced with EAP (Extensible Authentication Protocol). For instance, authentication can be based on a certificate and/or a user/password pair and outsourced to a RADIUS server.

WPA2. The latest version replaces RC4 by AES in CCM mode.

A.2 Block Chains

Bitcoin is a virtual currency which was released in 2009 by an anonymous person under the pseudonym *Satoshi Nakamoto*. It is based on a completely decentralized structure. There is no central bank. No inflation. No regulation. Anyone can create his own account and identify with an ECDSA public key. All transactions are public.

Transactions generate *unspent transaction outputs* (UTXO). The concept of *transaction* consists, for a user, in signing a message which says that he takes the few UTXO he owns and he spreads them among a list of users (which can include himself). This means that a transaction links to previous transactions (on which we shall collect the UTXO of the signer) and gives a list of shares associated to ECDSA public keys. These shares become the new UTXOs. Of course, the transaction is only valid if the sum of the shares is equal to the sum of the input UTXO and if these are valid UTXO. This raises the problem of identifying whether some value is really unspent. To check that, we need to check that among all past transactions, none has ever used this amount. This reduces to making sure that every user agrees on what is the list of all past transactions.

For that, transactions are published in a public ledger. Essentially, this is a chain of blocks. A block consists of the hash of the previous block (except for the genesis block), the list of all transactions since the ones in the last block, and a *proof-of-work*. Miners are collecting transactions and blocks. They consider as valid only the blocks which form the largest chain of blocks. Every period (which is of 10 minutes for bitcoins), they create a new block and broadcast it. There is a reward for the miner which makes a valid block.

The proof of work is such that the SHA256 hash of a block starts with many zeros. As of June 2016, the difficulty is to make them start with 69 zeros.

A.3 Mobile Telephony

The main security concern in mobile telephony is the authentication of the user. This is critical for the business model. Privacy is a secondary concern.

GSM. In the GSM architecture, users buy devices and subscribe to a home network. After subscription, they get a SIM card to be put inside the device and can connect to arbitrary networks. The SIM card and the home network share a symmetric key Ki which is critical. It is never given to the device nor to the visited network. This key is used to map a random challenge RAND to a response SRES and to an encryption key KC. When the mobile system connects to a visited network, it identifies itself so that the visited network can ask the home network for some triplets $(\mathsf{RAND}, \mathsf{SRES}, \mathsf{KC})$. Then, the network challenges the mobile system by sending the value of RAND. The SIM card computes SRES and KC and gives the result to the device. The device answers to the network with SRES. If it matches, the device is authenticated, and they can start communicating by encrypting using KC.

Prior to the authentication, the identification is done in clear in the first time when the device connects to the network. Then, the device and network synchronize on some temporary pseudonym which is sent encrypted by the network and used in clear the next time to identify. This is a pretty weak privacy protection.[1]

The algorithms to compute SRES (called A3) and KC (called A8) from Ki and RAND are only used by the SIM card and the home network. So, they do not need to be standard. Normally, the used algorithms are pretty good, but some weak ones such as COMP128 are circulating and careless operators may use them and deploy a poorly secured network: if a key recovery on A3 is possible, an adversary can send some chosen RAND challenges to the telephone to get the SRES in return over the air, and do the key recovery. If the key recovery on A8 is possible, a similar attack can be done by the telephone itself. If the key Ki is recovered, the SIM card can be cloned!

[1] Here, *privacy* does not refer to confidentiality. It refers to making the adversary unable to identify who is communicating.

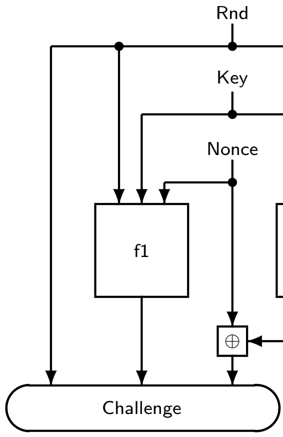

Figure A.1: MILENAGE Challenges in 3G Mobile Telephony

The KC-based encryption of the communication is done by all devices and visited networks. So, it must be standard. There are several options for encryption. (The option is imposed by the network to the device.) They can use A5/0, meaning that they don't use encryption. Otherwise, they can use A5/1, a weaker algorithm A5/2, or a stronger algorithm A5/3. Since fake networks can impose a weak encryption which is susceptible to key recovery attacks and that the key to be used is independent from the algorithm, the entire encryption system is weak.

We note that there is no integrity protection in the communication: the plaintext is just XORed to a keystream. Furthermore, replaying RAND forces to reuse the same keystream. So, message transmission has a weak security.

3G. In 3G mobile telephony, the challenges are authenticated and cannot be replayed. Furthermore, there is some integrity protection in messages (with a MAC) using another challenge-dependent symmetric key.

The challenges are protected using the MILENAGE scheme. Essentially, it comes from a secret key and a counter, which is authenticated and encrypted for privacy reasons. More precisely, there is a counter-based Nonce which is authenticated together with a non-repeating random value Rnd by a MAC-function f1 and which is encrypted with a stream cipher f5 using Rnd as a nonce (see, Fig. A.1). The challenge consists of Rnd, the authentication code, and the encrypted nonce. When the card receives the challenge, the nonce is decrypted and authenticated. Then, it is checked that the nonce is correct (e.g., based on the counter). So, the challenge can neither be forged nor reused.

The A3 algorithm is now replaced by an algorithm called f2 while A8 is replaced by two algorithms f3 and f4 to produce an encryption key and an authentication key, respectively. The overall architecture is depicted on Fig. A.2.

Encryption (f8) and MAC (f9) are done based on the KASUMI block cipher, using a special mode of operation.

The main security problem in 3G lies in the fact that networks are not authenticated to the mobile system. So, a fake network could still abuse the device. Furthermore, there is no encryption awareness: if the network (fake or not) imposes no encryption, the user is not aware of it.

A.4 Signal

Signal is a secure communication protocol which is used for instant messaging (it is used in WhatsApp). It was designed for secure messaging (confidentiality, integrity, and authentication)

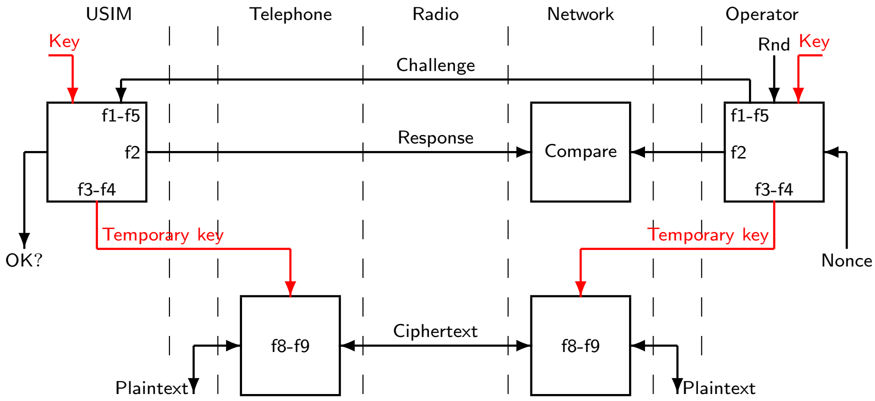

Figure A.2: The MILENAGE Protocol

and also with the notions of *forward secrecy*, *future secrecy*, and *plausible deniability* for stronger privacy enforcement.

Forward secrecy means that even though long-term secret may leak in the future, the confidentiality of current communication is preserved. Typically, we use Diffie-Hellman key agreement with ephemeral keys and erase these keys once the key agreement is done, so that they cannot leak. The ephemeral keys are authenticated with the long-term keys. So, leakage of long-term keys does not threaten confidentiality.

Future secrecy means that even though ephemeral secrets may leak, they do not compromise the secrecy of future communications. By doing frequent Diffie-Hellman key agreements, we are ensured that the leakage of ephemeral keys only affects communications within this session.

Plausible deniability means that no evidence will leak from communication. In particular, nothing is signed with a long-term key.

To achieve forward secrecy, Signal uses the concept of "*ratchet*". A ratchet is a mechanical device which can only move in only one direction. The key derivation of ephemeral keys should also be prevented to be stepped backward to achieve forward secrecy. Signal uses a double ratchet. One ratchet is based on Diffie-Hellman key agreement. For communication between Alice and Bob, every time the direction of communication changes, one Diffie-Hellman key agreement is done to derive a new secret key. In a second ratchet, this secret is updated using a one-way cryptographic function to derive a secret key for every new message, sent in the same direction.

To enforce plausible deniability, we use a triple Diffie-Hellman key agreement on the elliptic curve Curve25519. Essentially, given a long-term secret key a for Alice, a long-term public key bG for Bob, an ephemeral secret key $x_{a,\mathsf{eph}}$ for Alice, and an ephemeral public key $x_{b,\mathsf{eph}}G$ for Bob, Alice can compute three secrets $ax_{b,\mathsf{eph}}G$, $x_{a,\mathsf{eph}}bG$, and $x_{a,\mathsf{eph}}x_{b,\mathsf{eph}}G$. The only one depending on the long-term secret is $ax_{b,\mathsf{eph}}G$. It can be computed with an ephemeral secret $x_{b,\mathsf{eph}}$ as well, so this is no evidence which is binding for Alice. If we used abG, it would be binding as this can only be computed by Alice or Bob.

A.5 TLS

The TLS protocol (for *Transport Layer Security*) secures Internet exchanges: typically, the HTTP protocol. Several protocols are used, mostly the handshake protocol, to set up the cryptographic algorithms and a symmetric key, and the application data protocol which is the secure channel. The TLS communication is stateful. The state includes a session identifier, the peer certificate (if any), the selection of the cryptographic algorithms called cipher suite (if already selected), a master secret and nonces (if set up), and a sequence number to count the packets. The cipher suite includes an algorithm for peer authentication and key exchange, and a cipher spec which identifies a symmetric encryption and authentication algorithm.

In TLS 1.0, the algorithm for authentication and key exchange is mostly RSA (PKCS#1), but Diffie-Hellman with several variants (ephemeral, or semi-ephemeral) can also be used. Other algorithms have been added. If RSA is used, the handshake consists, for the server, in sending his RSA public key certificate. Then, the client selects a pre-master-secret, transfers it using the RSA public key of the server, and both the client and the server derive some secrets from it.

The cipher spec in TLS 1.0 include the NULL algorithm (doing nothing), DES, triple DES, DES with a 40-bit key, IDEA, RC4 (with 40-bit or 128-bit keys), and others, for encryption. Authentication is mostly done by algorithms called MD5 or SHA1. This means that HMAC based on these hash functions is used.

If the stream cipher RC4 is used, there is no nonce but the RC4 engine is never reset. This means that the state of the RC4 generator is kept in the session state for the next message to be encrypted. If a block cipher is used, it is with the CBC mode with a secret IV established during the handshake protocol.

The record protocol (to transmit data) uses the MAC-then-encrypt paradigm. If a block cipher is used, it is actually a MAC-then-pad-then-encrypt construction. A padding is done to obtain a integral sequence of blocks.

Both RC4 and the block ciphers have major security problems, these days. This will be fixed in the 1.3 version of TLS by moving to AES with a more decent authenticated encryption mode of operation.

The current draft of TLS 1.3 also includes changes in the key exchange algorithms. Now, the standard separates key exchange from peer authentication. Key exchange is exclusively done with ephemeral Diffie-Hellman, on an elliptic curve or in a field $\mathbf{Z}_p$. So, it provides *forward secrecy*. Authentication is done either with RSA certificates, or ECDSA certificates, or with the PSK method. PSK stands for *pre-shared key*. It consists in setting the `pre_master_key` to a key which was established in a previous session (this is called a *resumption*). Then, key derivation generates the session symmetric keys using HKDF. The encryption includes AES (in GCM or CCM mode) and the stream cipher CHACHA20. As for elliptic curves, TLS 1.3 requires secp256r1 to be implemented and also recommends X25519.

A.6 NFC Creditcard Payment

Credit cards mostly use the EMV payment standard. (EMV stands for Europay, Mastercard, and Visa.) Now, the wireless technology offers payment systems for credit cards. Those equipped with an NFC chip can now be used for contactless payment. Protocols are nearly the same but operate through the radio channel.

Essentially, when a *point of sale* (PoS) requests for payment, it asks the card to pay and the card does it without any action required from the holder on the card, as long as the radio contact with the card was made. First, the PoS gets the certificate for the public key of the card and extra identity information such as the serial number of the card (called PAN). Then, the PoS sends the amount to be paid and a nonce UN. The card then increments a transaction counter ATC and computes what is called a *cryptogram* AC and a signature SDAD. The cryptogram AC is actually a MAC computed on the amount to be paid, ATC, and information about the PoS. The MAC is based on a secret key which is shared with the bank. So, only the bank will understand the cryptogram. The signature SDAD signs AC, UN, the amount to pay, ATC, and information about the PoS. The signature can be verified by the PoS with the public key in the certificate. It can also be used to get the money if there is a problem with AC. Otherwise, only AC and the message authenticated by AC is sent to the bank to collect the payment.

Non-wireless payment with EMV are very similar. There are many variants and options though.

Clearly, the wireless technology can let the holder unaware that a payment is happening in his card. The amount paid by the card may be different to the amount he is willing to pay. Normally, the payment terminal is trusted to display the correct amount but this is only an act of faith. Next, the PoS is not authenticated. Finally, the protocol leaks some private information such as the identity of the card and its serial number. This is sent in clear through a radio channel to whoever asks for it to the card in a polite enough way (i.e., by following the standard protocol). Hence, we can easily imagine skimming attacks, in which malicious people would collect credit card numbers. We can also easily mount a relay attack to make people in the street pay in a remote shop.

Relay attacks are real threats to wireless access control system. They are being used to unlock cars and start the engine, to gain access to buildings or hotel rooms, to pay tolls on the highway, to pay with NFC whenever no PIN code is required, to access public transport, etc.

A.7 Bluetooth

Bluetooth is used to connect many sorts of wireless devices in a short-range network. There are several security modes. The standard one adds security on the link level. In this mode, devices can be *paired* with each other. This requires some input by the user. The default mode of a device is to be connectable but not visible: it does not answer to broadcast messages but only to messages sent to it. So, to be paired, it has to be put temporarily in a visible mode, where the devices can discover each other through the broadcast messages.

Paired devices share a long-term secret key. For each session, they authenticate in a challenge/response protocol using this key and derive session-dependent encryption keys. Again, the encryption is done by a stream cipher (with the synchronized clock to play the role of the nonce). So, there is no integrity protection for messages.

The legacy *pairing protocol* from the version 2.0 of Bluetooth distinguishes devices on which we can count on some memory and on a keyboard, from other (dummy) devices. On regular devices, the user must type an ephemeral PIN code on both devices. This PIN code is randomly selected by the user but must be the same on both devices. After that, the PIN code is not to be used anymore and can be discarded. Based on the PIN code, the two devices will exchange some nonces and derive the long-term symmetric key called the *link key*. On dummy devices, the PIN code is built in by the manufacturer and the user must type it on the other device. The dummy device does not want to have to store too many link keys (if it is to be paired with many other devices). So, it rather generates a unique *unit key* for himself when it is reset (e.g., switched on for the first time) and uses the pairing protocol to securely send his unit key to the other device. This implies that a third device could get the key controlling the security between the two devices by pairing with the dummy device.

More precisely, regular devices compute an ephemeral key K_{init} with an algorithm $\mathsf{E22}$ based on the PIN and a random number. This key is used to encrypt the two nonces in both directions. Finally, the two nonces are used to derive the long-term key K_{link} with the algorithm $\mathsf{E21}$ (see Fig. A.3). With a dummy device, there is no exchange of a nonce but the dummy device uses K_{init} to encrypt his long-term key K_{unit}, which is transmitted to the other device.

This pairing protocol is pretty weak when the PIN is short (which is often the case) and the adversary can listen to the communication during the pairing process [46]. If either the PIN is hard to find by exhaustive search or the communication is done in a safe place, the protocol is secure. It could eventually become secure by doing some frequent updates of the link key based on a previous one (which we call *repairing*), because it is unlikely that the adversary will be able to follow the updates in all protocols.

Encryption is done by a stream cipher called $\mathsf{E0}$ which is synchronous in the sense that the nonce which is used is the value of the clock register. So, devices must be synchronized. We note that encryption adds a linear redundancy check (CRC) which was originally aimed to protect the integrity but its security is void, as it was discussed in the case of the WEP security. So, communications can be corrupted.

Version 2.1. The 2.1 version adds some public-key cryptography techniques for pairing. This is called *Secure Simple Pairing (SSP)* protocol. This protocol distinguishes four association models. The ones offering resistance to active attacks are the *numeric comparison*, in which a human

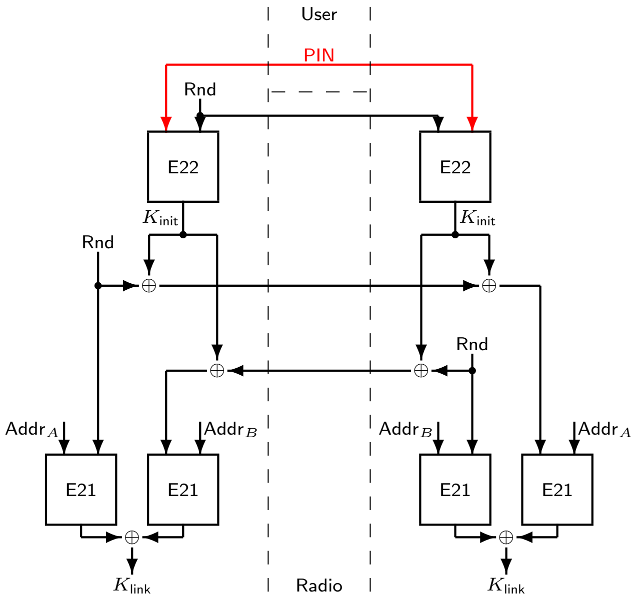

Figure A.3: The Bluetooth Legacy Pairing Protocol

operator must compare two numeric strings which play the role of a *Short Authenticated String* (SAS), and the *passkey entry*, in which the human user must enter an ephemeral secret (like in the legacy pairing). Two other techniques offer resistance to passive adversaries: *just works*, which is the same as the numeric comparison except that the comparison is not done, and *out-of-band*, which requires another (presumably) secure channel for key transmission.

For all variants, the two devices start with the Elliptic Curve Diffie-Hellman (ECDH) protocol to set up a secret DHKey. Then, they authenticate the ECDH execution in a way which depends on the association model. This phase also determines some nonces N_A and N_B, and some random values r_A and r_B. All this is fed to the algorithms $\mathsf{f3}$ and $\mathsf{f2}$ for checking and establishing the link key. The protocol is depicted on Fig. A.4.

In authentication stage 1, the numeric comparison model just consists of authenticating the ECDH exchanges by using the SAS. The protocol is depicted on Fig. A.5. The SAS is a numeric string which is displayed by both devices. The human operator tells both devices if the strings match (in the *numeric comparison* model), or this human comparison is skipped (in the *just works*

Device A		**Device** B
	$\boxed{\text{ECDH}}$	
DHKey		DHKey
	$\boxed{\text{authentication stage 1 (protocol dependent)}}$	
N_A, N_B, r_A, r_B		N_A, N_B, r_A, r_B
$E_A = \mathsf{f3}(\cdots)$	$\xrightarrow{E_A}$	check
check	$\xleftarrow{E_B}$	$E_B = \mathsf{f3}(\cdots)$
$\mathsf{LK} = \mathsf{f2}(\cdots)$		$\mathsf{LK} = \mathsf{f2}(\cdots)$
	$\boxed{\text{secure channel}}$	

Figure A.4: Bluetooth Secure Simple Pairing Protocol (SSP)

$$
\begin{array}{rcl}
\textbf{Device } A & & \textbf{Device } B \\
\textbf{input}\text{: } \mathsf{PK}_A, \widehat{\mathsf{PK}}_B & & \textbf{input}\text{: } \widehat{\mathsf{PK}}_A, \mathsf{PK}_B \\
& & \\
\text{pick } N_A \in_U \{0,1\}^{128} & & \text{pick } N_B \in_U \{0,1\}^{128} \\
\text{set } r_A = r_B = 0 & & \text{set } r_A = r_B = 0 \\
& \xleftarrow{\quad c_B \quad} & c_B \leftarrow \mathsf{f1}(\mathsf{PK}_B, \widehat{\mathsf{PK}}_A, N_B, 0) \\
& \xrightarrow{\quad N_A \quad} & \\
\hat{c}_B \stackrel{?}{=} \mathsf{f1}(\widehat{\mathsf{PK}}_B, \mathsf{PK}_A, \hat{N}_B, 0) & \xleftarrow{\quad N_B \quad} & \\
V_A \leftarrow g(\mathsf{PK}_A, \widehat{\mathsf{PK}}_B, N_A, \hat{N}_B) & & V_B \leftarrow g(\widehat{\mathsf{PK}}_A, \mathsf{PK}_B, \hat{N}_A, N_B) \\
\text{display } V_A & & \text{display } V_B \\
& \text{check } V_A = V_B & \\
& & \\
\textbf{output}\text{: } N_A, \hat{N}_B, r_A, r_B & & \textbf{output}\text{: } \hat{N}_A, N_B, r_A, r_B
\end{array}
$$

Figure A.5: Numeric Comparison Protocol for Authentication Stage 1 in SSP

model), in which case some active attacks are feasible. In the passkey entry model, the secret and the ECDH public keys are committed in both ways, then opened. This authenticates the ECDH keys. In the out-of-band model, another channel authenticates the commitments, in which the secret is set to a random value. Then, the commitments are opened in the regular channel.

In all cases, the idea is that we use ECDH with public keys authenticated in one way or another. ECDH uses the elliptic curve P192 from p. 43 and a KDF function of the x-coordinate.

The numeric comparison variant is quite interesting to look at more closely. Actually, it is based on a tricky interleave of a commitment c_B to N_B and the nonce N_A selection. (See Fig. A.5.) In this protocol, if the honest device A and B do not see the same public keys PK_A and PK_B, the adversary cannot make this protocol succeed. Indeed, he would have to make sure that $V_A = V_B$ for that. But to achieve $V_A = V_B$, he must control these values. However, the adversary cannot predict the value of V_B before he receives N_B from device B (because V_B is based on N_B which is hidden in the commitment), and the adversary cannot influence the value of V_A after he has sent c_B to device A (because the commitment makes sure he can opens it only to one N_B value, and the adversary does not know N_A when he releases c_B). So, $V_A = V_B$ occurs with small probability in that case.

Version 3.0. The 3.0 version makes Bluetooth compatible with WiFi by using a specific key derivation function to generate a specific link key to be used in this new channel. It is called Bluetooth High Speed (HS), and also AMP for Alternate MAC/PHY channel.

Version 4.0. The 4.0 version adds some protocol for sensors. It is called Bluetooth Low Energy (LE). It includes new key derivation and pairing protocols.

A.8 The Biometric Passport

ICAO, the UN organization standardizing passports, released in 2004 the first version of the MRTD (Machine Readable Travel Document) standard. This mandates that passports feature biometric recognition mechanism based on face recognition (mandatory), fingerprint or iris recognition (optional), with the support of a contactless chip. MRTD can be read by machines in two complementary ways: a Machine Readable Zone (MRZ) can be optically scanned, and they can communicate with the chip.

The MRZ includes basic information such as the name of the person, its date of birth, nationality, gender, the serial number of the passport, and its expiration date. The document number, date of birth, and expiration dates are elements which are used as a kind of password to run a (poor) password-based key exchange protocol and establish a secure channel. This is done by the (optional) *Basic Access Control (BAC)* protocol. Once done, triple DES and a variant of encrypted CBCMAC (EMAC) are used for the secure channel.

Several data groups are stored in the memory of the chip. The mandatory ones are the soft copy of the MRZ and a picture of the face. Another mandatory element contains the hash of all data groups to be authenticated, a signature of the list of digests, and sometimes a certificate for verifying the public key of the signature. This means that countries build their own public-key infrastructure. Passport issuing agencies have a certificate. The root self-signed certificate is authenticated by diplomatic protocols between countries. So, data groups are very securely authenticated. This authentication mechanism is called *passive authentication.* The memory of the chip also has room for many other data groups to contain private information such as fingerprint, iris scan, signature, address, profession, person to notify, electronic visas, travel records, etc.

An optional *Active Authentication (AA)* protocol can be used to prevent cloning attacks. With this protocol, the chip proves that it knows a secret key associated to a public key. The public key is one of the authenticated data groups, and the secret key never leaves the memory of the chip (i.e., it is not readable). This protocol is almost never implemented.

Recent versions of the standard now include an *Extended Access Control (EAC)* protocol. It includes PACE: a stronger protocol than BAC which is a PAKE based on the information from the MRZ. So, it is only vulnerable to a bruteforce online attack and provides forward secrecy. EAC also includes the Chip Authentication protocol which replaces AA. It also includes the Terminal Authentication protocol which authenticates the terminal based on an ECDSA certificate from the visited country. If this country is authorized, the reader can then read the non-mandatory data groups.

Overall, the data authentication of the MRTD technology is very secure (assuming that certificates are really verified, which may not always be the case). The BAC protocol leaks and provides poor security against unauthorized access to the chip but PACE and Terminal Authentication fixes it in EAC. Secure messaging is based on outdated algorithms (triple DES) that are still secure. Finally, the RFID technology which is used leaks private information. The usage of biometric data also leads to many types of threats.

Appendix B

The Shannon Entropy

Given a discrete random variable X, we define the *Shannon entropy* of X, that we denote by $H(X)$, the value

$$H(X) = -\sum_x \Pr[X = x] \log_2 \Pr[X = x]$$

Intuitively, $H(X)$ is the minimal number of bits of information that we need to encode X.

By extension, the *joint entropy* of a pair of random variables X and Y is the entropy or the variable (X, Y), i.e.

$$H(X,Y) = -\sum_{x,y} \Pr[X = x, Y = y] \log_2 \Pr[X = x, Y = y]$$

The *conditional entropy* $H(X|Y)$ is by definition

$$H(X|Y) = H(X,Y) - H(Y)$$

By manipulating the previous equations, we obtain

$$H(X|Y) = -\sum_{x,y} \Pr[X = x, Y = y] \log_2 \Pr[X = x|Y = y]$$

Intuitively, this is the number of bits to represent X once Y is already represented.

Lots of inequalities about the Shannon entropy come from the theory of convex functions. A real function f on the segment $[a, b]$ is said to be convex if and only if for every discrete set S, every function p from S to $]0, 1]$ such that $\sum_{s \in S} p_x = 1$ (that is a *weight function*), and every function t from S to $[a, b]$, we have

$$\sum_{x \in S} p_x f(t_x) \geq f\left(\sum_{x \in S} p_x t_x\right)$$

(p_x resp. t_x denote the image of x by the function p resp. t.) I.e., the weight sum of the $f(t_x)$'s is not smaller that the image of the weight sum of the t_x's. A function is further strictly convex if making the inequality an equality would always imply that all t_x's are equal. We know that for a function f which has a second derivative on $]a, b[$, f is strictly convex if and only if $f''(t) > 0$ for all $t \in]a, b[$.

Lemma B.1. *$H(X) \geq 0$ with equality if and only if X is constant.*

Proof. We define $f(t) = -\log_2 t$. Clearly, f is strictly convex on $[0, 1]$. We let S be the set of all x for which $\Pr[X = x] \neq 0$. By taking $t_x = p_x = \Pr[X = x]$, the definition of convexity gives that

$$H(X) \geq -\log_2\left(\sum_{x \in S} p_x^2\right)$$

Since $\sum_x p_x^2 \leq 1$, we obtain $H(X) \geq 0$.

Assuming equality, we must have $\sum_x p_x^2 = 1$ so all p_x must be equal to 1 so there must be a single term in the sum. I.e., S has a single point x. $\square$

Lemma B.2. *$H(X,Y) \geq H(X)$ with equality if and only if there exists some function f such that $Y = f(X)$ with probability 1.*

Proof. We first write

$$H(X,Y) - H(X) = H(Y|X) = \sum_x \Pr[X = x] \sum_y \Pr[Y = y|X = x] \log_2 \Pr[Y = y|X = x]$$

We look at the inner term in the sum over x. Given x fixed, due to Lemma B.1, the inner sum over y is non-negative. So, $H(X,Y) \geq H(X)$.

Now, if we have equality, all the sums over y must be null. Due to Lemma B.1, this implies that there exists a unique y (depending on x) for which $\Pr[Y = y|X = x] > 0$. We write it $y = f(x)$. Clearly, $\Pr[Y = f(x)|X = x] = 1$ for all x. So, we have $\Pr[Y = f(X)] = 1$. $\square$

Lemma B.3. *$H(X,Y) \leq H(X) + H(Y)$ with equality if, and only if X and Y are independent.*

Proof. The function $t \mapsto t \ln t$ has second derivative $\frac{1}{t}$. So, it is convex. By using the weights $\Pr[Y = y]$, we have

$$-\sum_y \Pr[Y = y] t_y \log_2 t_y \leq -\left(\sum_y \Pr[Y = y] t_y\right) \log_2 \left(\sum_y \Pr[Y = y] t_y\right)$$

with equality if and only if all t_y's for $\Pr[Y = y] \neq 0$ are equal. Applying this to $t_y = \Pr[X = x|Y = y]$ with x fixed yields

$$-\sum_y \Pr[X = x, Y = y] \log_2 \Pr[X = x|Y = y] \leq -\Pr[X = x] \log_2 \Pr[X = x]$$

with equality if and only if $\Pr[X = x|Y = y]$ does not depend on y. By summing over all x, we obtain $H(X|Y) \leq H(X)$ with equality if and only if X and Y are independent. $\square$

Lemma B.4. *If $\Pr[X = x] \neq 0$ for n values of x then $H(X) \leq \log_2 n$ with equality if, and only if all non-zero $\Pr[X = x]$ are equal to $\frac{1}{n}$.*

Proof. The function $t \mapsto -\ln t$ has second derivative $\frac{1}{t^2}$. So, it is convex. By using the weights $\Pr[X = x]$, we have

$$\sum_x \Pr[X = x] \log_2 t_x \leq \log_2 \left(\sum_x \Pr[X = x] t_x\right)$$

with equality if and only if all t_x's for $\Pr[X = x] \neq 0$ are equal. By applying this to $t_x = 1/\Pr[X = x]$, we obtain

$$H(X) \leq \log_2 n$$

with equality if and only if all nonzero $\Pr[X = x]$ are equal. $\square$

Bibliography

[1] Secure Hash Standard. *Federal Information Processing Standard* publication #180. U.S. Department of Commerce, National Institute of Standards and Technology, 1993. 6.3

[2] Secure Hash Standard. *Federal Information Processing Standard* publication #180-1. U.S. Department of Commerce, National Institute of Standards and Technology, 1995. 6.3, 6.3

[3] Secure Hash Standard. *Federal Information Processing Standard* publication #180-2. U.S. Department of Commerce, National Institute of Standards and Technology, 2002. 6.3

[4] Digital Signature Standard (DSS). *Federal Information Processing Standards* publication #186-2. U.S. Department of Commerce, National Institute of Standards and Technology, 2000. 6.2, 7.4

[5] Advanced Encryption Standard (AES). *Federal Information Processing Standards* Publication #197. U.S. Department of Commerce, National Institute of Standards and Technology, 2001. 5.2

[6] Data Encryption Standard (DES). *Federal Information Processing Standard* publication #46–3. U.S. Department of Commerce, National Institute of Standards and Technology, 1999. 5.2

[7] The XTS-AES Tweakable Block Cipher: The XTS-AES Tweakable Block Cipher. IEEE Std 1619-2007. Institute of Electrical and Electronics Engineers, 2008. 5.2

[8] PKCS#1v1: RSA Cryptography Standard. An RSA Laboratories Technical Report Version 1.5. RSA Laboratories, 1993. 7.3

[9] PKCS#1v2.1: RSA Cryptography Standard. RSA Security Inc. Public-Key Cryptography Standards. RSA Laboratories, 2002. 7.3, 7.3

[10] M. Abdalla, D. Pointcheval. Simple Password-Based Encrypted Key Exchange Protocols. In *Topics in Cryptology CT-RSA'05*, San Fransisco, California, U.S.A., Lecture Notes in Computer Science 3376, pp. 191–208, Springer-Verlag, 2005. 8.2

[11] D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J.A. Halderman, N. Heninger, D. Springall, E. Thomé, L. Valenta, B. VanderSloot, E. Wustrow, S.Z. Béguelin, P. Zimmermann. Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice. In *22nd ACM Conference on Computer and Communications Security*, Denver CO, USA, pp. 5–17, ACM Press, 2015. 2.6

[12] S.S. Al-Riyami, K.G. Paterson. Certificateless Public Key Cryptography. In *Advances in Cryptology ASIACRYPT'03*, Taipei, Taiwan, Lecture Notes in Computer Science 2894, pp. 452–473, Springer-Verlag, 2003. 8.6

[13] G. Avoine, S. Vaudenay. How to Safely Close a Discussion. *Information Processing Letters*, vol. 102, pp. 138–142, 2007. 8.3

[14] M. Bellare. New Proofs for NMAC and HMAC: Security without Collision-Resistance. In *Advances in Cryptology CRYPTO'06*, Santa Barbara, California, U.S.A., Lecture Notes in Computer Science 4117, pp. 602–619, Springer-Verlag, 2006. 6.4

[15] S.M. Bellovin, M. Merritt. Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attacks. In *IEEE Symposium on Research in Security and Privacy*, Oakland, California, USA, pp. 72–84, IEEE Computer Society Press, 1992. 8.2

[16] C.H. Bennett. Logical Reversibility of Computation. *IBM Journal of Research and Development*, vol. 17, pp. 525–532, 1973. 5.6

[17] D.J. Bernstein, T. Lange. Non-Uniform Cracks in the Concrete: the Power of Free Precomputation. In *Advances in Cryptology ASIACRYPT'13*, Bengaluru, India, Lecture Notes in Computer Science 8270, pp. 321–340, Springer-Verlag, 2013. 4.6

[18] . D.J. Bernstein. Curve25519: New Diffie-Hellman Speed Record. In *Public Key Cryptography'06*, New York NY, USA, Lecture Notes in Computer Science 3958, pp. 207–228, Springer-Verlag, 2006. 4.6

[19] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche. The Keccack Reference v3.0. 2011. `http://keccak.noekeon.org/` 6.3

[20] J. Black, P. Rogaway. A Block-Cipher Mode of Operation for Parallelizable Message Authentication. In *Advances in Cryptology EUROCRYPT'02*, Amsterdam, Netherlands, Lecture Notes in Computer Science 2332, pp. 384–397, Springer-Verlag, 2002. 6.4

[21] D. Boneh. Simplified OAEP for the RSA and Rabin Functions. In *Advances in Cryptology CRYPTO'01*, Santa Barbara, California, U.S.A., Lecture Notes in Computer Science 2139, pp. 275–291, Springer-Verlag, 2001. 7.3

[22] D. Boneh, X. Boyen. Short Signatures Without Random Oracles. In *Advances in Cryptology EUROCRYPT'04*, Interlaken, Switzerland, Lecture Notes in Computer Science 3027, pp. 56–73, Springer-Verlag, 2004. 4.8, 7.4

[23] D. Boneh, X. Boyen. Short Signatures Without Random Oracles and the SDH Assumption in Bilinear Groups. *Journal of Cryptology*, vol. 08, pp. 149–177, 2008. 4.8, 7.4

[24] D. Boneh, M.K. Franklin. Identity-Based Encryption from the Weil Pairing. In *Advances in Cryptology CRYPTO'01*, Santa Barbara, California, U.S.A., Lecture Notes in Computer Science 2139, pp. 213–229, Springer-Verlag, 2001. 4.8, 8.6, 8.6

[25] D. Boneh, B. Lynn, H. Shacham. Short Signatures from the Weil Pairing. *Journal of Cryptology*, vol. 17, pp. 297–3192, 2004. 4.8, 7.4

[26] N. Borisov, I. Goldberg, D. Wagner. Intercepting Mobile Communications: the Insecurity of 802.11. In *International Conference on Mobile Computing and Networking MOBICOM'01*, Rome, Italy, pp. 180–189, ACM, 2001. A.1

[27] E. Brickell, D. Pointcheval, S. Vaudenay, M. Yung. Design Validations for Discrete Logarithm Based Signature Schemes. In *Public Key Cryptography'00*, Melbourne, Australia, Lecture Notes in Computer Science 1751, pp. 276–292, Springer-Verlag, 2000. 7.4

[28] J.L. Carter, M.N. Wegman. Universal Classes of Hash Functions. *Journal of Computer and System Sciences*, vol. 18, pp. 143–154, 1979. 6.4

[29] F. Chabaud, A. Joux. Differential Collisions in SHA-0. In *Advances in Cryptology CRYPTO'98*, Santa Barbara, California, U.S.A., Lecture Notes in Computer Science 1462, pp. 56–71, Springer-Verlag, 1998. 6.3

[30] R. Cramer, V. Shoup. Design and Analysis of Practical Public-Key Encryption Schemes Secure against Adaptive Chosen Ciphertext Attack. *SIAM Journal on Computing*, vol. 33, pp. 167–226, 2003. 7.8

[31] I.B. Damgård. A Design Principle for Hash Functions. In *Advances in Cryptology CRYPTO'89*, Santa Barbara, California, U.S.A., Lecture Notes in Computer Science 435, pp. 416–427, Springer-Verlag, 1990. 6.3

[32] W. Diffie, M.E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, vol. IT-22, pp. 644–654, 1976. 2.2, 2.7, 2.8, 7.1

[33] H. Dobbertin. Cryptanalysis of MD4. *Journal of Cryptology*, vol. 11, pp. 253–271, 1998. 6.3

[34] A. Duc, S. Vaudenay. TCHo: a Code-based Cryptosystem. In *Advances in Network Analysis and its Applications (E. Kranakis Ed.)*, Mathematics in Industry 18, pp. 149–180, Springer, 2013. 7.7

[35] M. Dworkin. Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC. NIST Special Publication 800-38D. U.S. Department of Commerce, National Institute of Standards and Technology, 2007. 6.4

[36] M.J. Dworkin. SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. *Federal Information Processing Standards* publication #202. U.S. Department of Commerce, National Institute of Standards and Technology, 2015. 6.3, 6.3

[37] T. ElGamal. A Public-key Cryptosystem and a Signature Scheme based on Discrete Logarithms. *IEEE Transactions on Information Theory*, vol. IT-31, pp. 469–472, 1985. 2.8, 7.2, 7.4

[38] R.W. Floyd. Nondeterministic Algorithms. *Communications of the ACM*, vol. 14, pp. 636–644, 1967. 6.6

[39] E. Fujisaki, T. Okamoto. Secure Integration of Asymmetric and Symmetric Encryption Schemes. In *Advances in Cryptology CRYPTO'99*, Santa Barbara, California, U.S.A., Lecture Notes in Computer Science 1666, pp. 537–554, Springer-Verlag, 1999. 7.7

[40] E. Fujisaki, T. Okamoto. Secure Integration of Asymmetric and Symmetric Encryption Schemes. *Journal of Cryptology*, vol. 26, pp. 80–101, 2013. 7.7

[41] S. Goldwasser, S. Micali. Probabilistic Encryption and How to Play Mental Poker Keeping Secret All Partial Information. In *Proceedings of the 14th ACM Symposium on Theory of Computing*, San Fransisco, California, U.S.A., pp. 365–377, ACM Press, 1982. 3.4

[42] S. Goldwasser, S. Micali. Probabilistic Encryption. *Journal of Computer and System Sciences*, vol. 28, pp. 270–299, 1984. 3.4

[43] L. Grover. A Fast Quantum Mechanical Algorithm for Database Search. In *Proceedings of the 28th ACM Symposium on Theory of Computing*, Philadelphia, Pennsylvania, U.S.A., pp. 212–219, ACM Press, 1996. 5.6, 6.8

[44] J. Hoffstein, J. Pipher, J.H. Silverman. NTRU: A Ring Based Public Key Cryptosystem. In *Algorithmic Number Theory — Proceedings of ANTS-III*, 1423, Lecture Notes in Computer Science Portland, Oregon, U.S.A., pp. 267–288, Springer-Verlag, 1998. 7.7

[45] T. Iwata, K. Kurosawa. OMAC: One-Key CBC MAC. In *Fast Software Encryption'03*, Lund, Sweden, Lecture Notes in Computer Science 2887, pp. 129–153, Springer-Verlag, 2003. 6.4

[46] M. Jakobsson, S. Wetzel. Security Weaknesses in Bluetooth. In *Topics in Cryptology CT-RSA'01*, San Fransisco, California, U.S.A., Lecture Notes in Computer Science 2020, pp. 176–191, Springer-Verlag, 2001. A.7

[47] A. Joux. A One-Round Protocol for Tripartite Diffie-Hellman. *Journal of Cryptology*, vol. 17, pp. 263–276, 2004. 4.8

[48] A. Joux, V. Vitse. Elliptic Curve Discrete Logarithm Problem over Small Degree Extension Fields. *Journal of Cryptology*, vol. 26, pp. 119–143, 2013. 4.4, 4.6

[49] J. Kim, A. Biryukov, B. Preneel, S. Hong. On the Security of HMAC and NMAC Based on HAVAL, MD4, MD5, SHA-0 and SHA-1. In *Security and Cryptography for Networks SCN'06*, 4116, Lecture Notes in Computer Science Maiori, Italy, pp. 242–256, Springer-Verlag, 2006. 6.4

[50] J. Kohl, C. Neuman. The Kerberos Network Authentication Service (V5). Internet standard. RFC 1510, 1993. 8.6

[51] H. Krawczyk. LFSR-based Hashing and Authentication. In *Advances in Cryptology CRYPTO'94*, Santa Barbara, California, U.S.A., Lecture Notes in Computer Science 839, pp. 129–139, Springer-Verlag, 1994. 6.3, 6.4

[52] H. Krawczyk, M. Bellare, R. Canetti. HMAC: Keyed-Hashing for Message Authentication. RFC 2104, the Internet Society, 1997. 6.4

[53] R. Landauer. Irreversibility and Heat Generation in the Computing Process. *IBM Journal of Research and Development*, vol. 5, pp. 183–191, 1961. 5.6

[54] H.W. Lenstra. Factoring Integers with Elliptic Curves. *Annals of Mathematics*, vol. 126, pp. 649–673, 1987. 3.5

[55] A.K. Lenstra. Key Lengths. In *The Handbook of Information Security* (H. Bidgoli Editor). John Wiley & Sons 2004. 7.5

[56] A.K. Lenstra, H.W. Lenstra. *The Development of the Number Field Sieve*, Springer-Verlag, 1993. 2.6, 3.5

[57] C.H. Lim, P.J. Lee. A Study on the Proposed Korean Digital Signature Algorithm. In *Advances in Cryptology ASIACRYPT'98*, Beijing, China, Lecture Notes in Computer Science 1514, pp. 175–186, Springer-Verlag, 1998. 7.4

[58] R.J. McEliece. A Public-Key Cryptosystem based on Albegraic Coding Theory. In *DSN Progress Report 42–44*, Jet Propulsion Laboratory, Pasadena, California, U.S.A., pp. 114–116, 1978. 7.7

[59] R.C. Merkle. One Way Hash Functions and DES. In *Advances in Cryptology CRYPTO'89*, Santa Barbara, California, U.S.A., Lecture Notes in Computer Science 435, pp. 416–427, Springer-Verlag, 1990. 6.3

[60] M. Naor, S. Shamir. Visual Cryptography. In *Advances in Cryptology EUROCRYPT'94*, Perugia, Italy, Lecture Notes in Computer Science 950, pp. 1–12, Springer-Verlag, 1995. 1.4

[61] K. Nyberg, R. Rueppel. A New Signature Scheme Based on the DSA Giving Message Recovery. In *1st ACM Conference on Computer and Communications Security*, Fairfax VA, U.S.A., pp. 58–61, ACM Press, 1993. 7.4

[62] T. P. Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *Advances in Cryptology CRYPTO'91*, Santa Barbara, California, U.S.A., Lecture Notes in Computer Science 576, pp. 129–140, Springer-Verlag, 1992. 7.8

[63] S. Pohlig, M. Hellman. An Improved Algorithm for Computing Logarithms over GF(q) and its Cryptographic Significance. *IEEE Transactions on Information Theory*, vol. IT-24, pp. 106–110, 1978. 2.6

[64] J. M. Pollard. A Monte Carlo Method for Factorization. *Nordisk Tidskrift for Informations-behandlung (BIT)*, vol. 15, pp. 331–334, 1975. 4.6

[65] C. Pomerance. A Tale of Two Sieves. *Notices American Mathematical Society*, vol. 43, pp. 1473–1485, 1996. 2.6

[66] M.O. Rabin. Digitalized Signatures and Public-Key Functions as Intractable as Factorization. Technical Report MIT/LCS/TR–212, MIT Laboratory for Computer Science, 1979. 7.3

[67] O. Regev. On Lattices, Learning with Errors, Random Linear Codes, and Cryptography. *Journal of the ACM*, vol. 56(6), 2009. 7.7

[68] E. Rescorla. Diffie-Hellman Key Agreement Method. RFC 2631, the Internet Society, 1999. 2.7

[69] R.L. Rivest. The MD4 Message Digest Algorithm. In *Advances in Cryptology CRYPTO'90*, Santa Barbara, California, U.S.A., Lecture Notes in Computer Science 537, pp. 303–311, Springer-Verlag, 1991. 6.3

[70] R.L. Rivest. The MD5 Message Digest Algorithm. RFC 1321, the Internet Society, 1992. 6.3

[71] R.L. Rivest, A. Shamir and L.M. Adleman. A Method for Obtaining Digital Signatures and Public-key Cryptosystem. *Communications of the ACM*, vol. 21, pp. 120–126, 1978. 3.1, 3.1, 3.3, 7.2, 7.3

[72] A. Sahai, B. Waters. Fuzzy Identity-Based Encryption. In *Advances in Cryptology EU-ROCRYPT'05*, Aarhus, Denmark, Lecture Notes in Computer Science 3494, pp. 457–473, Springer-Verlag, 2005. 4.8

[73] C.P. Schnorr. Efficient Identification and Signature for Smart Cards. In *Advances in Cryptology CRYPTO'89*, Santa Barbara, California, U.S.A., Lecture Notes in Computer Science 435, pp. 235–251, Springer-Verlag, 1990. 7.4

[74] C.P. Schnorr. Efficient Identification and Signature for Smart Cards. *Journal of Cryptology*, vol. 4, pp. 161–174, 1991. 7.4

[75] P. Sepehrdad, P. Sušil, S. Vaudenay, M. Vuagnoux. Smashing WEP in A Passive Attack. In *Fast Software Encryption'13*, Singapore, Lecture Notes in Computer Science 8424, pp. 155–178, Springer-Verlag, 2013. A.1

[76] D. Shanks. Class Number, a Theory of Factorization and Genera. In *Symposium in Pure Mathematics*, Providence, R.I., pp. 415-—440, AMS, 1971. 2.6

[77] C.E. Shannon. Communication Theory of Secrecy Systems. *Bell system technical journal*, vol. 28, pp. 656–715, 1949. 1.4

Reedited by N. J. A. Sloane and A. D. Wyner in *Claude Elwood Shannon collected papers*, IEEE Press, 1993.

[78] P.W. Shor. Algorithms for Quantum Computation: Discrete Logarithms and Factoring. In *Proceedings of the 35th IEEE Symposium on Foundations of Computer Science*, Santa Fe, New Mexico, U.S.A., pp. 124–134, IEEE, 1994. 2.6, 3.5, 7.7

[79] J.H. Song, J. Lee, T. Iwata. The AES-CMAC Algorithm. RFC 4493, the Internet Society, 2006. 6.4

[80] R. Solovay, V. Strassen. A fast Monte-Carlo Test for Primality. *SIAM Journal on Computing*, vol. 6, pp. 84–86, 1977. 3.4

[81] M. Stevens, E. Bursztein, P. Karpman, A. Albertini, Y. Markov. The First Collision for Full SHA-1. In *Advances in Cryptology CRYPTO'17*, Santa Barbara, California, U.S.A., Lecture Notes in Computer Science 10401–10403, pp. 570–596 vol. 1, Springer-Verlag, 2017. 6.3

[82] S. Vaudenay. Secure Communications over Insecure Channels Based on Short Authenticated Strings. In *Advances in Cryptology CRYPTO'05*, Santa Barbara, California, U.S.A., Lecture Notes in Computer Science 3621, pp. 309–326, Springer-Verlag, 2005. 8.5

[83] G.S. Vernam. Cipher Printing Telegraph Systems for Secret Wire and Radio Telegraphic Communications. *Journal of the American Institute of Electrical Engineers*, vol. 45, pp. 109–115, 1926. 1.4, 6.4

[84] X. Wang, X. Lai, D. Feng, H. Chen, X. Yu. Cryptanalysis of the Hash Functions MD4 and RIPEMD. In *Advances in Cryptology EUROCRYPT'05*, Aarhus, Denmark, Lecture Notes in Computer Science 3494, pp. 1–18, Springer-Verlag, 2005. 6.3

[85] X. Wang, Y.L. Yin, H. Yu. Finding Collisions in the Full SHA-1. In *Advances in Cryptology CRYPTO'05*, Santa Barbara, California, U.S.A., Lecture Notes in Computer Science 3621, pp. 17–36, Springer-Verlag, 2005. 6.3

[86] X. Wang, H. Yu. How to Break MD5 and Other Hash Functions. In *Advances in Cryptology EUROCRYPT'05*, Aarhus, Denmark, Lecture Notes in Computer Science 3494, pp. 19–35, Springer-Verlag, 2005. 6.3

[87] X. Wang, H. Yu, W. Wang, H. Zhang, T. Zhan. Cryptanalysis on HMAC/NMAC-MD5 and MD5-MAC. In *Advances in Cryptology EUROCRYPT'09*, Cologne, Germany, Lecture Notes in Computer Science 5479, pp. 121–133, Springer-Verlag, 2009. 6.4

[88] X. Wang, H. Yu, Y.L. Yin. Efficient Collision Search Attacks on SHA-0. In *Advances in Cryptology CRYPTO'05*, Santa Barbara, California, U.S.A., Lecture Notes in Computer Science 3621, pp. 1–16, Springer-Verlag, 2005. 6.3

[89] M.N. Wegman, J.L. Carter. New hash functions and their use in authentication and set equality. *Journal of Computer and System Sciences*, vol. 22, pp. 265–279, 1981. 6.4

[90] D. Whiting, R. Housley, N. Ferguson. Counter with CBC-MAC (CCM). RFC 3610, the Internet Society, 2003. 6.4

[91] R.S. Winternitz. Producing a One-Way Hash Function from DES. In *Advances in Cryptology CRYPTO'83*, Santa Barbara, California, U.S.A., pp. 203–207, Plenum Press, 1983. 6.3

[92] T. Wu. Telnet Authentication: SRP. Request for Comments RFC 2944, The Internet Society, 2000. 8.2