Paolo Dolce

A concise introduction to cryptography

June 20, 2020

Contents

Introduction

Cryptography is a very ancient discipline and was born to satisfy two opposite purposes: the privacy protection of a private conversation and on the other hand the eavesdropping of a private conversation.
The situation is modelled in the following way: Alice and Bob are friends situated in different places of the world and Alice wants to send some secret messages to Bob. There is also Eve that is an eavesdropper and tries to intercept the messages.

Remark 0.1. These notes are a very short introduction to cryptography by the point of view of Alice and Bob, so the main aim of our reasonings is to present several ways to ensure secure conversations. In other words here we are on Alice and Bob's side, they are the good guys, whereas, Eve is the enemy.

We suppose that Eve is a very smart and clever person, so she can always intercept the communication. Thus Alice and Bob, in order to protect the privacy, must hide the real significance of the message, adopting a sort of enciphered language. The most naive way to construct this type of communication is the following: let's indicate with $\mathscr{M}$ the set of all messages (or plaintexts) that can be sent; for every plaintext $m \in \mathscr{M}$, Alice encrypts it creating the cryptogram (or chiphertext) $\mathtt{e}(m)$. The set of all cryptograms is denoted by $\mathscr{C}$, and $\mathtt{e} : \mathscr{M} \longrightarrow \mathscr{C}$ is the encrypting function. Bob receives the cryptogram $c := \mathtt{e}(m)$ and decrypts it with a certain decrypting function $\mathtt{d} : \mathscr{C} \longrightarrow \mathscr{M}$ such that $\mathtt{d}(c) = m$.
This naive scheme is useless because we always assume that Eve has the following capabilities:

(E1) She can access to an arbitrarily large number of ciphertexts, in other words she intercepts all the communications between Alice and Bob.
(E2) She always knows the encrypting and decrypting functions.

Remark 0.2. The assumption (E1) and (E2) are quite realistic. Indeed one can imagine that Alice and Bob are being wiretapped and that the encrypting and decrypting functions are well known by the experts.

The assumption (E2) seems very strong, but it can be easily overcome if Alice and Bob use a secret key k to decrypt every ciphertext. In this way $\mathtt{e}$ and $\mathtt{d}$ become functions of two variables: m is encrypted as $c := \mathtt{e}(m, k)$ and it is recovered by applying $\mathtt{d}(c, k)$. With this trick, even if Eve knows $\mathtt{e}$ and $\mathtt{d}$, she still needs the secret key k, so at least we are sure that she can't find the plaintext "in the obvious way".

Remark 0.3. The encrypting function $\mathtt{e}$ is in general necessary for communications, even in presence of a secret key k. In fact delivering a "huge message" can be a real nuisance, therefore $\mathtt{e}$ is often a compression function too.

We now have the informal skeleton of the encryption/decryption scheme, that is a cryptosystem. From here there are basically two ways to proceed for the formal construction of a secure cryptosystem: one is the symmetric cryptography and the other is the modern cryptography. We will explain the difference between these two approaches, but we point out that the entire philosophy whereby Alice and Bob face up Eve is completely different between the two ways.
Finally, in order to underline how is difficult to maintain the secrecy, we describe some sneaky attacks that Eve can perform:

- *Known plaitext attack* (KPA): Eve has some samples of the plaintext and its encrypted version.

- *Chosen plaintext attack* (CPA): Eve has the capability to choose arbitrary plaintexts to be encrypted and obtain the corresponding ciphertexts.

Remark 0.4. The chosen plaintext attack could seem a bit unreal, in fact it would certainly be unlikely that Eve (who is the enemy) could persuade Alice to encrypt large amounts of plaintexts for her. By the way we are not overestimating Eve's capabilities, indeed she, for example, could hack the Alice's device for sending messages.

Chapter 1
Symmetric cryptography

The philosophy behind symmetric cryptography is very simple and naive: Alice and Bob share a set of secret keys and each secret key is used for hiding the messages to Eve (the adjective "symmetric" it refers to the fact that both the guys have the secret keys). Clearly the set of secret keys and their usage must be agreed before the starting of the communications.

1.1 Symmetric cryptosystems and perfect secrecy

We start with the formal definition of a symmetric cryptosystem:

Definition 1.1. A *symmetric cryptosystem* is a collection

$$\big(\mathscr{M}, \mathscr{C}, \mathscr{K}, \mathtt{e}(\cdot,\cdot), \mathtt{d}(\cdot,\cdot)\big)$$

where:

- $\mathscr{M}$ is the (finite) set of messages (or plaintexts), $\mathscr{C}$ is the (finite) set of cryptograms (or ciphertexts), and $\mathscr{K}$ is the (finite) set of secret keys.
- $\mathtt{e} : \mathscr{M} \times \mathscr{K} \longrightarrow \mathscr{C}$ and $\mathtt{d} : \mathscr{C} \times \mathscr{K} \longrightarrow \mathscr{M}$ are respectively the encrypting and decrypting functions with the following property: $\mathtt{d}(\mathtt{e}(m,k),k) = m$, for every secret key $k \in \mathscr{K}$ and for every plaintext $m \in \mathscr{M}$.

In this context, Eve sees the ciphertext $c = \mathtt{e}(m,k)$ and she knows the functions $\mathtt{e}$ and $\mathtt{d}$ but, to obtain m she can't compute $\mathtt{d}(c,k)$ like Bob does because she doesn't know the secret key k. Therefore Eve can't recover the plaintext m in a obvious way.

Proposition 1.2. *For any symmetric cryptosystem* $|\mathscr{M}| \leq |\mathscr{C}|$

Proof. The function $\mathtt{e}(\cdot,k)$ with k fixed is injective since $\mathtt{d}(\cdot,k)$ is its left inverse, so $|\mathscr{M}| \leq |\mathscr{C}|$. □

We have not precise data about Eve's capabilities or knowledge, maybe she is a friend of Alice and Bob therefore she may have some partial informations about the secret key, or, for some reasons, she is able to restrict the field of search for the plaintext. These, are situations that we can't completely control in a deterministic way, but we can model them thanks to the probability theory. Given a symmetric cryptosystem $\big(\mathscr{M}, \mathscr{C}, \mathscr{K}, \mathtt{e}(\cdot,\cdot), \mathtt{d}(\cdot,\cdot)\big)$, consider the two measurable spaces $(\mathscr{M}, 2^{\mathscr{M}})$, $(\mathscr{K}, 2^{\mathscr{K}})$ and a generic probability space $(\Omega, \mathcal{H}, P)$, then we define two independent random variables:

$$M : \Omega \longrightarrow \mathscr{M}$$

$$K : \Omega \longrightarrow \mathscr{K}$$

where $P(M = m)$ represents the probability that the message m is sent and $P(K = k)$ represents the probability that the key k is used. Note that the distributions of the random variables M and K capture the informations holding to Eve, in fact by her point of view, some messages or some keys can be "more probable" than others.

Remark 1.3. We always suppose that $P(M = m) > 0$ for every $m \in \mathscr{M}$ and $P(K = k) > 0$ for every $k \in \mathscr{K}$.

Considering moreover the measurable space $(\mathscr{C}, 2^{\mathscr{C}})$, it is evident that the function $\mathtt{e}$ is measurable, so $C := \mathtt{e}(M, K)$ is a random variable and

$$C : \Omega \longrightarrow \mathscr{C}$$

where $P(C = c)$ is the probability that the ciphertext c is received. Thanks to the independence of M and K we obtain that

$$P(C = c) = P(\omega \in \Omega \,:\, \mathtt{e}(M(\omega), K(\omega)) = c) =$$

$$= P\left(\bigsqcup_{(m,k)\in \mathtt{e}^{-1}(c)} M = m, K = k\right) \;= \sum_{(m,k)\in \mathtt{e}^{-1}(c)} P(M = m)P(K = k)$$

Remark 1.4. Possibly restricting $\mathscr{C}$ to the image of the function $\mathtt{e}$ we can always suppose that $P(C = c) > 0$ for every c, this means we will always consider only cryptograms that can be effectively received, namely from now the encrypting function $\mathtt{e}$ will be surjective.

Roughly speaking we say that a symmetric cryptosystem is perfectly secret if Eve doesn't obtain any more information about the plaintext when she intercepts the ciphertext; in other words the encrypted message doesn't provide any further hint to the eavesdropper. Remember that a perfectly secret symmetric cryptosystem doesn't ensure that the plaintexts will never be discovered from the eavesdropper, in fact Eve, thanks to her knowledge of Alice and Bob could however try to guess the plaintext with some kind of criterion. We are only saying that intercepting or not the ciphertext is indifferent for the Eve's purposes.
Now we present in a succint and simplified way the concept of perfect secrecy introduced by Shannon in 1949 (the original reference is [Sha49]) :

Definition 1.5. Consider a symmetric cryptosystem $\big(\mathscr{M}, \mathscr{C}, \mathscr{K}, \mathtt{e}(\cdot,\cdot), \mathtt{d}(\cdot,\cdot)\big)$, we say that it is *perfectly secret* if M and C are two independent random variables.

Proposition 1.6. *A symmetric cryptosystem is perfectly secret if and only if for every plaintext $m \in \mathscr{M}$ and every cryptogram $c \in \mathscr{C}$ we have that:*

$$P(M = m \,|\, C = c) = P(M = m)$$

Proof. The classical formula of conditioned probabilities is

$$P(M = m \,|\, C = c) = \frac{P(M = m, C = c)}{P(C = c)}$$

therefore $P(M = m \,|\, C = c) = P(M = m)$ if and only if M and C are indipendent. □

Thus, with the perfect secrecy we ask the probability that m is the sent message, conditioned by the fact that some ciphertext c is received, is equal to the initial probability that m is sent.

Proposition 1.7. *For any perfectly secret symmetric cryptosystem it holds that $|\mathscr{M}| \leq |\mathscr{C}| \leq |\mathscr{K}|$.*

Proof. The first inequality was proven in proposition 1.2 so we need to show only that $|\mathscr{C}| \leq |\mathscr{K}|$. Fix a message m_0, then thanks to the perfect secrecy hypothesis, $P(M = m_0 \,|\, C = c) = P(M = m_0) > 0$ for all ciphertext c; so there is at least a key $k \in \mathscr{K}$ such that $\mathtt{e}(m_0, k) = c$. This implies the map $\mathtt{e}(m_0, \cdot)$ is surjective so $|\{m_0\} \times \mathscr{K}| = |\mathscr{K}| \geq |\mathscr{C}|$. □

The above proposition expresses in some sense "the cost" of the perfect secrecy, indeed to obtain it we need a huge number of keys, at least as many as cryptograms. This is a severe drawback of the perfect secrecy: suppose in fact that Alice sends n messages (in different instants of time) to Bob with n large enough. Since $P(K = k) > 0$, by the Borel's law of large numbers all the keys in the set $\mathscr{K}$ will be used by Alice, so Bob to decrypt all the ciphertexts should have an enormous list like the following

- The key for the message number 1 is k_1
- The key for the message number 2 is k_2

  $\vdots$
- The key for the message number n is k_n

where each k_i is randomly chosen and $\bigcup_{i=1}^{n} k_i = \mathscr{K}$. This list can be obviously agreed in secrecy before the communications for example with a physical meeting, but first of all it puts a bound to the number of messages that Alice can send to Bob, and moreover if it were found by Eve, the security of many future plaintexts would be irrimediably compromised.

1.2 Two simple examples of symmetric cryptosystems

We now present two well known cryptosystems.

Mono-alphabetic substitution

This is an ancient cryptosystem (also Giulio Cesare used it for his private communications) that permits to transmit words of any language. The idea si very simple: every letter of the plaintext is encrypted as another letter through a permutation of the alphabet in use. Suppose that Alice and Bob communicate in English, so the set $\mathscr{M}$ is made by all english words and the set of secret keys is $\mathscr{K} = \{\pi\}$ with $\pi \in S_{26}$ (with the notation S_{26} we indicate the symmetric group over 26 elements). We associate to each letter of the English alphabet a number from 1 to 26 according to the natural sorting, thus `A` $= 1$, `B` $= 2$, $\ldots$, `Z` $= 26$ and we can assume that a plaintext m is a sequence $m = (\alpha_1, \ldots, \alpha_t)$ with $\alpha_i \in \{1, \ldots, 26\}$. Finally we define the encrypting and decrypting functions:

$$\mathtt{e}((\alpha_1, \ldots, \alpha_\ell), \pi) := (\pi(\alpha_1), \ldots, \pi(\alpha_\ell))$$

$$\mathtt{d}((\beta_1, \ldots, \beta_\ell), \pi) := (\pi^{-1}(\beta_1), \ldots, \pi^{-1}(\beta_\ell))$$

We obtain a symmetric cryptosystem called the mono-alphabetic substitution cipher. Since we have only a key and a plenty of messages, it is clearly not perfectly secret due to the proposition 1.6. Mono-alphabetic substitution is very easy to break through a frequency analysis: some letters of the English alphabet are more frequent than others, for example `E`, `T` and `A` are the most frequent and `X`, `J` and `Z` are the less frequent. Since Eve can access to a consistent number of cryptograms, she enumerates all the intercepted letters and then she replaces the most common letter with an `E`, the second most common with a `T` and the third most common with a `A`. In similar way the three least common letters are substituted with `X`, `J` and `Z`. At this point the second step is to analyze also the frequency of bigrams a trigrams in English language to make more substitutions. Now, with some simple reasoning on the partially guessed plaintext, it is possible to find the sent message and the secret key π; moreover, once Eve finds the key π she can decrypt all future cypertexts. By the way note that with a chosen plaintext attack, Eve can find the secret key almost immediately without any frequency analysis. We conclude that the mono-alphabetic substitution is a very insecure cryptosystem since it can be easily broken in different ways.

One-time pad

We now present the "unbreakable cryptosystem" for excellence. Alice encodes the information she wants to send with the ASCII code, bounding the length of the encoded messages with a large constant $\ell = 8n$ (n bytes). Therefore every message is sent as a bit string of length ℓ with the convention that if the encoded plaintext's dimension is less than ℓ, the remaining space is filled with zeros. We have that $\mathscr{M} = \Sigma^\ell$, where $\Sigma = \{0, 1\}$, and moreover we set $\mathscr{K} = \mathscr{C} = \mathscr{M}$. The encrypting and decrypting functions are the followings:

$$\mathsf{e}(m,k) = m \oplus k$$

$$\mathsf{d}(c,k) = c \oplus k$$

where the operation $\oplus$ is simply the sum modulo 2 componentwise (i.e. for example $10010111 \oplus 01100011 = 11110100$). With the above setup $(\mathscr{M}, \mathscr{C}, \mathscr{K}, \mathsf{e}, \mathsf{d})$ is a symmetric cryptosystem called one-time pad.

Proposition 1.8. *Consider the one-time pad cryptosystem, if the random variable K is uniformly distributed then the cryptosystem is perfectly secret.*

Proof. We know that

$$P(C=c) = \bigsqcup_{(m,k)\in \mathsf{e}^{-1}(c)} P(M=m, K=k) \;=\; \sum_{(m,k)\in \mathsf{e}^{-1}(c)} P(M=m)P(K=k) \tag{1.1}$$

but $\mathsf{e}^{-1}(c) = \{(m,k) \,:\, m \in \mathscr{M},\, k = m \oplus c\}$, namely, fixed $c \in \mathscr{C}$, for every $m \in \mathscr{M}$ there is only a key $k = m \oplus c \in \mathscr{K}$ such that $\mathsf{e}(m, m \oplus c) = c$. Since $P(K=k) = \frac{1}{2^\ell}$ for every $k \in \mathscr{K}$, by equation (2.1) we get:

$$P(C=c) = \sum_{m\in\mathscr{M}} \frac{P(M=m)}{2^\ell} = \frac{1}{2^\ell}$$

On the other hand it is obvious that

$$P(C=c|M=m) = P(K = m \oplus c) = \frac{1}{2^\ell}$$

therefore we can now calculate $P(M=m|C=c)$ for any $m \in \mathscr{M}$ and $c \in \mathscr{C}$:

$$P(M=m|C=c) = \frac{P(M=m, C=c)}{P(C=c)} = \frac{P(C=c|M=m)P(M=m)}{P(C=c)} = \\ = \frac{2^{-\ell}P(M=m)}{2^{-\ell}} = P(M=m)$$

□

Remark 1.9. The crucial point is that any key k must be chosen in a completely random way amongst all possible 2^ℓ strings, otherwise we can't ensure the perfect secrecy.

The one-time pad is absolutely unbreakable, even if Eve uses a chosen plaintext attack, but it has a terrible disadvantage we will immediately explain. If the length of the messages ℓ is very large also the key wherewith the message is encrypted must be of length ℓ by construction of the cryptosystem. So if Alice needs to send many big messages to Bob, both the guys must generate and manage an enormous ordered list of very long keys, but this is not very feasible in reality.

The perfect secrecy is nowadays considered only an ideal tool, namely a "cryptographers dream" that is unacheveable for real applications. So, at this point we need to change completely the philosophy, and find other tools to ensure the security of the messages. There are essentially two ways to proceed: the first is to insist on symmetric cryptography but with different perspectives, and the second is to modify completely the framework, passing to a modern concept of cryptography
We will explore only the second way, but we point out that there also many efficient symmetric cryptosystems still used like: LFSR type ciphers, DES or AES (for a brief overview of this cryptosystems see for example [TW06]).

Chapter 2
Modern cryptography

With the evolution of hardware's technology, the humankind has obtained an immense computational power, but despite this, still today, it is conjectured the existence of some problems for which, even if the existence of a solution is known, it can't be calculated in a reasonable amount time. The (conjectural) existence of such problems is at the base of modern cryptography; in fact the "new setup" is the following: we require that Alice can encrypt *efficiently* the messages, but on the other hand, for Eve must be computationally infeasible to recover the plaintexts. To be more precise, even if Eve constructs some algorithms to recover the sent messages, this algorithms are certainly *not efficient* in the sense that they need an enormous amount of time to perform the task. Note the fundamental difference with the arguments of the previous chapters, in fact we don't care about the information carried by the ciphertexts, we "only" have to ensure that even if Eve knows how to recover the plaintexts, she cant effectively make the computations. We concentrate our attention on Eve's computational capabilities and not on her knowledge or smartness. Therefore it is natural to base modern cryptography on complexity theory. The first thing that needs to be clarified is the following: What is the computational limit of the three guys? We will answer to this question in section 2.1. Moreover in sections 2.2, 2.3 and 2.4 we will formalize the construction of a scheme that ensures easy encryptions but severe difficulties for the eavesdropper's purposes. The extraordinary feature of this approach is that we can create some secure cryptosystems that don't need any key sharing, so we can avoid the necessity of "previous agreements" between Alice and Bob. Finally, In the last two sections we will present two widely used modern cryptosystems.

2.1 A survey on some probabilistic complexity classes

We know from basic computational complexity theory that a problem can be efficiently solved if there exists a TM (Turing machine) that solve it in a polynomial time of the input length (if the encoding scheme is reasonable). This statement is not entirely true, in fact in the real world there are many problems not yet solved with polynomial time algorithms, but considered, however, efficiently solved.
We also know that a NTM (non deterministic Turing machine) has much more power than a TM, but unfortunately it represents an ideal model of calculus not entirely reproducible in reality. However we can effectively produce a special NTM that is a middle way between a TM and a NTM, that is a PTM (probabilistic Turing machine).

Definition 2.1. A *probabilistic Turing machine* (PTM) is a multi-tape TM M which has two different transition functions δ_0 and δ_1 such that, in performing each computational step, M uses the function δ_i with probability $p_i = \frac{1}{2}$ for $i = 1, 2$.

It is very easy to see that a PTM is a NTM, infact if $M = (\Gamma, Q, \delta_0, \delta_1)$ is a PTM where Γ is the alphabet and Q the state-set, then we have the relation

$$\big\{(q, \gamma, q', \gamma', \mathfrak{m}) \subseteq (\Gamma \times Q) \times \big(\Gamma \times \overline{Q} \times \{\leftarrow, -, \rightarrow\}\big) : (q', \gamma', \mathfrak{m}) = \delta_i((q, \gamma))\big\}$$

for some $i \in \{0,1\}$. NTMs and PTMs are however two basically different objects; firstly the former have a generic computation tree, insted the latter has a computation tree with vertexes of outgoing valence 0 or 2. Moreover the computational choices of PTMs are probabilized and at each step this choice is made by a fair-coin tossing, whereas for NTMs we have not informations on the non-deterministic choice. Another way to visualize a PTM is thinking it as a multi-tape TM where a specified only-read tape contain a very long random sequence of bits that clearly changes for every input. In this way the current state of the machine depends on the random bit in reading.

Remark 2.2. By possibly interrupting the machine if the running time is too large, we can assume that on every input x, every computational branch of a PTM eventually halts.

Consider a PTM M that halts on some input x using a computational branch b of depth t (with the letter b we always will indicate a computational branch), then the probability that M "choses b" is clearly $\Pr(b) = \frac{1}{2^t}$ (note that in this chapter the probability function is indicated with $\Pr(\cdot)$ and not with $P(\cdot)$). Moreover, for any input x, the number of steps required to halt is $t_{M,x}$; it clearly is a random variable depending on the computational branch chosen by the machine and we have

$$\Pr(t_{M,x} = k) = \frac{|\{\text{branches of depth } k\}|}{|\{\text{branches}\}|}$$

We indicate with $T_{M,x}$ the maximum value attained by the random variable $t_{M,x}$, namely the maximum length for a computational branch of $M(x)$.

Definition 2.3. A PTM M has probabilistic running time $f(n)$, where $f : \mathbb{N} \longrightarrow \mathbb{N}$, if

$$f(n) = \max_{x \in \Sigma^n} T_{M,x}$$

When f is a polynomial, we say that M ha probabilistic polynomial running time.

Definition 2.4. A PTM M has expected running time $g(n)$, where $g : \mathbb{N} \longrightarrow \mathbb{N}$, if

$$g(n) = \max_{x \in \Sigma^n} E[t_{M,x}]$$

When g is a polynomial we say that M has polynomial expected running time.

Remark 2.5. Roughly speaking the expected running time is the everange running time of a PTM.

Definition 2.6. The probability that a PTM M halts with the state $\mathtt{H}$ on an input x is:

$$\Pr(M(x) = \mathtt{H}) := \sum_{\substack{b \text{ that leads to} \\ M(x)=\mathtt{H}}} \Pr(b)$$

Finally we fix the limit for the computational capabilities of the three guys:

Claim. Alice, Bob and Eve can perfom only those algorithms that, translates as PTMs, have at most polinomial running time.

We conclude the section with a panoramic view of the most important complexity classes for decisional problem, thus we assume for simplicity that our PTMs have only the two halt states `YES` and `NO`. However what follows is not strictly necessary for understanding cryptography.

Definition 2.7. Consider a language Λ and a PTM M, then for any $\epsilon \in \mathbb{R}$ such that $0 \leq \epsilon \leq \frac{1}{2}$ we say that:

- M accepts Λ with error ϵ if $\Big(x \in \Lambda\Big) \Rightarrow \Big(\Pr(M(x) = \mathtt{YES}) \geq 1 - \epsilon\Big)$
- M decides Λ with error ϵ if $\Big(x \in \Lambda\Big) \Rightarrow \Big(\Pr(M(x) = \mathtt{YES}) \geq 1 - \epsilon\Big)$ and moreover $\Big(x \notin \Lambda\Big) \Rightarrow \Big(\Pr(M(x) = \mathtt{NO}) \geq 1 - \epsilon\Big)$

Practically M decides Λ with error ϵ if the PTM gives the right answers about the belonging of an element in Λ with probability $1-\epsilon$.
Now we want to define the polynomial time-complexity for a PTM relative to a certain error:

Definition 2.8. The set $\mathbf{BPTIME}(f(n))$ (**BP-** stands for bounded-error probabilistic) contains all languages Λ decided by a PTM with error $\epsilon = \frac{1}{4}$ and in time $O(f(n))$. The complexity class **BPP** (bounded-error probabilistic polynomial) is defined as:

$$\mathbf{BPP} := \bigcup_{c\in\mathbb{N}} \mathbf{BPTIME}(n^c)$$

Remark 2.9. From the definition it is evident that $\mathbf{P} \subseteq \mathbf{BPP}$, but on the contrary it is an open problem to prove or disprove that $\mathbf{P} = \mathbf{BPP}$. So we don't formally know if probabilistic polynomial algorithms are really more powerful than polynomial algorithms. Moreover it is not known the relation between **BPP** and **NP**.

At first glance the choice of $\epsilon = \frac{1}{4}$ is a bit unfortunate, in fact an algorithm that goes wrong with probability $\frac{1}{4}$ is practically useless. This is not a real problem, because thanks to the proposition below, if $\Lambda \in \mathbf{BPP}$, then there exists a PTM that decides Λ with an error ϵ arbitrarily close to 0 and in polynomial time:

Proposition 2.10. *If $\Lambda \in \boldsymbol{BPP}$, then for any polynomial $f : \mathbb{N} \longrightarrow \mathbb{N}$, there exists a PTM that decides Λ with an error $\epsilon = 2^{-f(n)}$ and in polynomial time.*

Proof. When $f(n) = 1$ or $f(n) = 2$ there is nothing to prove, so we can suppose that $f(n) \geq 3$. If M is the PTM that decides Λ with error $\epsilon = \frac{1}{4}$ and in a polynomial time, then we construct another PTM M' in the following way: on any input x with $|x| = n$, M' executes $2t+1$ times the machine M on x, where $t \in \mathbb{N}$ (this number will be determined later in the proof), and records the array $v \in \{\texttt{YES}, \texttt{NO}\}^{2t+1}$ containing all the halt states of M. If at lest $t+1$ entries of v are `YES`, then $M'(x) = \texttt{YES}$ otherwise $M'(x) = \texttt{NO}$. If $x \in \Lambda$, then the probability that $M'(x) = \texttt{NO}$ is given by:

$$\begin{aligned}
\Pr(M'(x) = \texttt{NO}) &= \Pr(M(x) = \texttt{YES} \text{ for at most } t \text{ times}) \leq \\
&\leq \sum_{k=0}^{t} \binom{2t+1}{k} \left(\frac{3}{4}\right)^k \left(\frac{1}{4}\right)^{2t+1-k} \leq \\
&\leq (t+1)\binom{2t+1}{k} \left(\frac{3}{4}\right)^t \left(\frac{1}{4}\right)^{t(n)+1} = \\
&= \frac{t+1}{4}\binom{2t+1}{k} \left(\frac{3}{16}\right)^t \leq \frac{t+1}{4} 2^{2t+1} \left(\frac{3}{16}\right)^t = \\
&= (t+1)\frac{2^{2t-1}}{2^{2t}} \left(\frac{3}{4}\right)^t \leq (t+1)\left(\frac{3}{4}\right)^t
\end{aligned}$$

Now if we put $t = 8f(n)$, since $\left(\frac{3}{4}\right)^8 \leq \frac{1}{8}$ we have that

$$(8f(n)+1)\left(\frac{3}{4}\right)^{8f(n)} \leq (8f(n)+1)\left(\frac{1}{8}\right)^{f(n)} = 2^{-f(n)}\left[(8f(n)+1)\,4^{-f(n)}\right]$$

But for $f(n) \geq 3$ it is easy to show that $(8f(n)+1)\,4^{-f(n)} \leq 1$ so we can finally conclude that

$$\Pr(M'(x) = \texttt{NO}) \leq 2^{-f(n)}$$

By symmetry, in the identical way one shows that if $x \notin \Lambda$, then

$$\Pr(M'(x) = \texttt{YES}) \leq 2^{-f(n)}$$

Since $2t+1$ and $f(n)$ are two polynomials, in both cases it is evident that the PTM M' runs for a polynomial time. □

Remark 2.11. From the above proposition it is clear that the class **BPP** could have been defined using as error any ϵ such that $0 \leq \epsilon < \frac{1}{2}$.

What we have just proven is very strong, in fact even if we operate with a polynomial PTM, only by repeating its task polynomial many times, we can obtain an error arbitrarily close to 0. It means that, despite the decisional problems contained in **BPP** are solved in a non deterministic way, they are "extremely close" from being decisional problems that lie in **P**. A PTM that decides language $\Lambda \in \mathbf{BPP}$ is also called a PTM with *two-sided* error; this because it can halt with the wrong state when either $x \in \Lambda$ or $x \notin \Lambda$. For some kinds of problems, we need other types of algorithms, more precisely we need *one-sided* error PTMs and *zero-sided* error PTMs. We therefore introduce other two probabilistic computational classes that captures this ideas of unilateral or absent error.

Definition 2.12. The set $\mathbf{RTIME}(f(n))$ (here **R-** stands for randomized) contains all languages Λ such that there exists a PTM with running time $O(f(n))$ and with the following properties:

- $\Big(x \in \Lambda\Big) \Rightarrow \Big(\Pr(M(x) = \texttt{YES}) \geq \frac{1}{2}\Big)$
- $\Big(x \notin \Lambda\Big) \Rightarrow \Big(\Pr(M(x) = \texttt{NO}) = 1\Big)$

The complexity class **RP** is defined as

$$\mathbf{RP} := \bigcup_{c \in \mathbb{N}} \mathbf{RTIME}(n^c)$$

In other words **RP** contains those languages accepted with error $\frac{1}{2}$ in polynomial time and such that the PTM never goes wrong if $x \notin \Lambda$. In this case the constant $\frac{1}{2}$ of the definition can be replaced with every number $0 \leq \epsilon < 1$:

Proposition 2.13. *If $\Lambda \in \boldsymbol{RP}$, then for any polynomial $f : \mathbb{N} \longrightarrow \mathbb{N}$ there exists a PTM M' that on every input $x \in \Sigma^n$ runs in polynomial time and moreover has the following properties:*

- $\Big(x \in \Lambda\Big) \Rightarrow \Big(\Pr(M'(x) = \mathit{YES}) \geq 1 - 2^{-f(n)}\Big)$
- $\Big(x \notin \Lambda\Big) \Rightarrow \Big(\Pr(M'(x) = \mathit{NO}) = 1\Big)$

Proof. Suppose that M is the PTM that accepts Λ in the sense of the previous definition, then M' is simply the PTM that simulates $f(n)$ times the machine M on input x and records the array $v \in \{\texttt{YES}, \texttt{NO}\}^{f(n)}$. If at least an entry of v is YES then $M'(x) = \texttt{YES}$, otherwise $M'(x) = \texttt{NO}$. So when $x \in \Lambda$ the probability that $M'(x) = \texttt{YES}$ is:

$$\Pr(M'(x) = \texttt{YES}) =$$

$$= \Pr(M(x) = \texttt{YES} \text{ at least once amongst } f(n) \text{ times}) \geq 1 - 2^{-f(n)}$$

The other case is trivial. □

Remark 2.14. Obviously from the definitions we have that $\mathbf{P} \subseteq \mathbf{RP} \subseteq \mathbf{NP}$ and moreover thanks to the proposition 2.13 we can conclude that $\mathbf{RP} \subseteq \mathbf{BPP}$.

Both **BPP** and **RP** represent a set of decisional problems that can be solved in a wrong way by a PTM although the probability of error could be very small. We now introduce a third complexity class that contains all decisional problems solved with no error by a PTM, but with the drawback that a few computational branch of this PTM can be of exponential depth. Practically in this case, when a PTM runs on an input x, even if we know that at the end we will obtain the right answer, it can happens that we have to wait an enormous amount of time to see the halt state.

Definition 2.15. The set $\mathbf{ZPTIME}(f(n))$ (**Z-** stands for zero-error) contains all languages Λ decided by a PTM without any error (so $\epsilon = 0$) and with expected running time $O(f(n))$. The complexity class **ZPP** (zero-error probabilistic polynomial) is defined as:

$$\mathbf{ZPP} := \bigcup_{c \in \mathbb{N}} \mathbf{ZPTIME}(n^c)$$

Theorem 2.16. *The following equality holds:* $\boldsymbol{ZPP = RP \cap co\text{-}RP}$

Proof. ($\subseteq$) Consider a language $\Lambda \in \mathbf{ZPP}$, then there is a PTM that decides it without errors and with expected running time $g(n) = O(f(n))$. We construct the PTM that simulates M but that halts `NO` when the running time exceeds $2g(n)$ steps (practically we cut out the too long branches of the computation graph of M). If $x \notin \Lambda$, then we have always $M'(x) = \texttt{NO}$ with no errors, conversely if $x \in \Lambda$ we are interested to the probability $\Pr(t_{M,x} > 2g(n))$ and thanks to the Markov's inequality[1] we have that

$$\Pr\left(t_{M,x} \geq 2g(n)\right) \leq \frac{E[t_{M,x}]}{2g(n)} = \frac{1}{2}$$

It follows that $\Pr(M'(x) = \texttt{YES}) \geq \frac{1}{2}$, therefore $\Lambda \in \mathbf{RP}$. We can also construct a PTM M'' that simulates M but that halts `YES` when the running time exceeds $2g(n)$ and argue in the same way as above that $\Lambda \in \mathbf{co\text{-}RP}$. ($\supseteq$) Suppose that $\Lambda \in \mathbf{RP} \cap \mathbf{co\text{-}RP}$, then there are two PTMs M and N that operates respectively with running times $f_M(n)$ and $f_N(n)$ and with the following properties:

- If M halts `YES` then $x \in \Lambda$
- If N halts `NO` then $x \notin \Lambda$

We now construct the PTM M' as follows: on any input x, M'' simulates repeatedly first M and then N on the input x until we obtain $M(x) = \texttt{YES}$ or $N(x) = \texttt{NO}$. Clearly M'' never goes wrong and moreover

$$E[t_{M'',x}] \leq \sum_{k=0}^{\infty} \frac{k+1}{2^k} \max\{f_M(n), f_N(n)\} = 4 \max\{f_M(n), f_N(n)\}$$

namely $\mathbf{RP} \cap \mathbf{co\text{-}RP} \subseteq \mathbf{ZPP}$. □

Remark 2.17. The above theorem and the remark 2.14 yield the following sequence of complexity classes $\mathbf{P} \subseteq \mathbf{ZPP} \subseteq \mathbf{RP} \subseteq \mathbf{BPP}$.

2.2 One-way functions

First of all we specify what we mean by "inverting" a function:

Definition 2.18. Let $f : S \longrightarrow \Sigma^*$ be a calculable function. For every $x \in S$ consider the set

$$f^{-1}(f(x)) = \{z \in \Sigma^* \ : \ f(z) = f(z)\}$$

We say that f is invertible if for every $x \in S$ there exists a TM M such that $M(f(x)) \in f^{-1}(f(x))$

So in this context to invert a function it is enough to find a counter image of every $f(x)$ with an algorithm. It is very easy to find functions easy to be calculated but hard to be inverted, look the following example.

Example 2.19. The function $f : \Sigma^* \longrightarrow \Sigma^*$, where $\Sigma = \{0, 1\}$, such that

$$f(x) = \text{least } \lfloor \log_2(|x|) \rfloor \text{ bits of } x$$

can be obviously calculated in polynomial time, but if z is a counter image of x, then $|z| \geq 2^{\lfloor |f(x)| \rfloor}$, so can't exist a TM that inverts f in polynomial time.

As the above example shows, a function can be hard to invert only because the counter images of every x have exponential length respect to $|f(x)|$. In cryptography we don't want to consider such functions as one-way functions, because in this case "the hardness" lies only in writing down a counter image but not in finding it.

Definition 2.20. A function $f : S \subseteq \Sigma^* \longrightarrow \Sigma^*$ is *honest* if for every y in $f(S)$ and every counter image $x \in f^{-1}(y)$ there exists a polynomial $q : \mathbb{N} \longrightarrow \mathbb{N}$ such that $|x| \leq q(|y|)$.

[1] The Markov's inequality says that if $X \geq 0$ is a random variable and $t > 0$, then $\Pr(X \geq t) \leq \frac{E[X]}{t}$

Remark 2.21. A honest function doesn't shrink exponentially the strings of its domain.

Therefore, if f is honest and it comes out that f is hard to invert, "the hardness" consists exclusively in finding a counter image. We arrive at the formal definition of a one-way function:

Definition 2.22. A function $f : \Sigma^{\leq n} \subseteq \Sigma^* \longrightarrow \Sigma^*$, where $\Sigma = \{0, 1\}$, is a *one-way function* if the following conditions hold:

1) f is honest
2) f can be calculated in deterministic polynomial time.
3) If M is any PTM that runs in polynomial time, then for every polynomial $q : \mathbb{N} \longrightarrow \mathbb{N}$ there is a $k_q \in \mathbb{N}$ such that for every k with $k_q \leq k \leq n$ we have that

$$\Pr\left(M\left(f(U_k)\right) \in f^{-1}(f(U_k))\right) \leq \frac{1}{q(k)} \tag{2.1}$$

where U_k is the random variable with uniform distribution over Σ^k and the probability is taken over the distribution of U_k and over the non deterministic computation of M.

Definition 2.23. if f is a one-way function and moreover it is a permutation of Σ^n then is called *one-way permutation*.

Remark 2.24. In the definition 2.22, the condition 3) becomes completely different if we replace equation (2.1) with the following one that at first glance may seem more friendly:

$$\Pr\left(M\left(f(x)\right) \in f^{-1}(x))\right) \leq \frac{1}{q(k)} \quad \forall x \in S \tag{2.2}$$

In fact such a condition would be too strong and the set of one-way founction would be certainly empty. To see this, suppose that $f : \Sigma^{\leq n} \longrightarrow \Sigma^*$ is a generic function and fix a generic $x_0 \in S$; then consider the TM G such that $G(y) = x_0$ for every $y \in \Sigma^*$. Clearly G operates in deterministic polynomial time and moreover if $y = f(x_0)$ then $G(f(x_0)) = x_0$, so we conclude that on the specific input x_0, the machine G inverts f with probability 1 (no matter what is f). On the other hand

$$\Pr\left(G\left(f(U_k)\right) \in f^{-1}(f(U_k))\right) =$$

$$= \Pr\left(f(x_0) = f(U_k)\right) = \Pr\left(U_k \in f^{-1}(f(x_0))\right)$$

so, if for example f is in injective, this probability is less than $\frac{1}{2^k}$. Practically G is a "guessing TM", indeed it tries to invert f claiming that the inverse of a generic string $f(x)$ is a fixed element x_0. When f is not highly non-injective this algorithm has poor chances to succeed if x is chosen at random, but if x is chosen in $f^{-1}(f(x))$, then G never goes wrong.

In other words the point 3) of definition 2.22 says that if Eve tries to invert a one-way function *on a random input* x and with all her computational capabilities, the probability she succeeds is very small. Practically the inversion of f fails *almost always* with any probabilistic polynomial time algorithm.

Definition 2.25. A function $g : \mathbb{N} \longrightarrow \mathbb{N}$ is said *negligible* if for every polynomial $q : \mathbb{N} \longrightarrow \mathbb{N}$ there exists a number $k_q \in \mathbb{N}$ such that for every $k > k_q$ it holds that $g(k) \leq \frac{1}{q(k)}$.

Remark 2.26. The condition 3) of definition 2.22 can be restated as follows

3)′ If M is any PTM that runs in polynomial time, the function

$$m \longmapsto \Pr\left(M\left(f(U_m)\right) \in f^{-1}(f(U_m))\right)$$

is negligible.

Once we have the formal definitions, it is natural to ask if does exist at least a one-way function, but in order to answer it is crucial the following theorem.

Theorem 2.27. *If there exists a one-way function then* $\mathbf{P} \neq \mathbf{NP}$.

Proof. We show that if $\mathbf{P} = \mathbf{NP}$, then any function f that is honest and computable in polynomial time can be inverted in deterministic polynomial time.
Consider the language

$$L_f = \{x \sqcup y \in \Sigma^* \cup \{\sqcup\} : \exists z \in \Sigma^* \text{ s.t. } |z| = q(|xy|) \text{ and } f(xz) = y\}$$

(q is a polynomial), and let M_f be the TM that operates in the following way: on input $(x \sqcup z \sqcup y)$, $M(x \sqcup y \sqcup z) = \texttt{YES}$ if $f(xz) = y$, and $M(x \sqcup y \sqcup z) = \texttt{NO}$ otherwise. Since f is calculable in polynomial time, it means that for every possible input we have a certificate respect to L_f, namely $L_f \in \mathbf{NP}$. Now look at the algorithm 1, it that takes as input a generic string $y := f(x)$ and returns a counter image $u \in f^{-1}(y)$. By

Input: $y := f(x)$
$u \leftarrow 0$;
if $u \sqcup y \in L_f$ **then**
  ;
  else
    $u = 1$;
  end
end
while $f(u) \neq y$ **do**
  if $u0 \sqcup y \in L_f$ **then**
    $u \leftarrow u0$;
    else
      $u \leftarrow u1$;
    end
  end
end
Output: $u \in f^{-1}(y)$

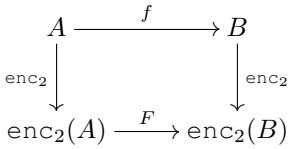
Algorithm 1: Inversion of f

the hypothesis $\mathbf{P} = \mathbf{NP}$ it follows that $L_f \in \mathbf{P}$, and moreover, since f is honest, the while cycle runs for a polynomial number of steps. We can conclude that the algorithm 1 has polynomial running time, and the proof is complete. □

It is clear that we don't know if there exist a one-way function, since if it existed, the problem $\mathbf{P}$ vs $\mathbf{NP}$ would be already solved. However in the next section are showed some good candidates for being one-way functions, and we will conjecture that they are really one-way functions.

2.3 Candidates for one-way functions

We present some functions that are generally assumed to be one-way. They all play a foundamental role in modern cryptography and they are closely related to easy number theory considerations. From now, we will adopt the following convention: when a function $f : A \longrightarrow B$ is given between two number sets, the associated function F between strings is simply "the traduction of f in binary". That is if $\texttt{enc}_2$ is the encoding function of numbers in bit strings, then $F : \texttt{enc}_2(A) \longrightarrow \texttt{enc}_2(B)$ is a function such that the following diagram commutes:



Factorization

This is the oldest and most widely studied candidate for being a one-way function. Let p and q be two prime numbers and then consider the trivial mutiplication function $\texttt{mult}(p,q) = pq$. The corresponding function $\texttt{Mult}$ between bit strings, whose input is the concatenation of the two primes, is clearly honest and calcolable in deterministic polynomial time. But can $\texttt{Mult}$ be easily inverted? To be more precise, given a natural number x that is a product of two primes how is hard to find its factorization? The most naive algorithm in order to solve this problem is a brute force algorithm that tries direct divisions starting from the number 2 up to $\lfloor\sqrt{x}\rfloor$. Note that it is enough to search a divisor in the interval $\{2,\dots,\lfloor\sqrt{x}\rfloor\}$ since

$$\frac{x}{\lfloor\sqrt{x}\rfloor + n} < \sqrt{x} \quad \forall n \in \mathbb{N}\setminus\{0\}$$

so a factor of x is found before arriving to $\lfloor\sqrt{x}\rfloor + 1$. Each trial division can be performed in polynomial time, but in the worst case we need $\lfloor\sqrt{x}\rfloor - 1$ divisions, that is an exponential (with base 2) number in fuction of the input length. We conclude that the brute force algorithm has exponential running time.
After many years of search for a feasible algorithm to invert $\texttt{Mult}$, the best one is the general number field sieve (GNFS) algorithm with running time $O\left(e^{\frac{64|x|}{9}^{\frac{1}{3}}+\ln(|x|)^{\frac{2}{3}}}\right)$. This incapability, since today, to find a polynomial time (even probabilistic) algorithm leads to the following well accepted assumption:

Conjecture 2.28. (Factoring assumption) The function $\texttt{Mult}$ is a one-way function.

Discrete logarithm

Fix a prime number p and a generator b of the group $\mathbb{Z}_p^*$, then the group homomorphism

$$\begin{aligned}\texttt{dexp}_{(p,b)} : \mathbb{Z} &\longrightarrow \mathbb{Z}_p^* \\ x &\longmapsto [b^x]_p\end{aligned}$$

is surjective. Now consider the associated function between strings $\texttt{dExp}_{(p,b)}$ that takes as inputs the concatenation of p, b, and x (encoded in binary) and outputs the binary representation of $b^x \pmod p$. We see that $\texttt{dExp}_{(p,b)}$ is calculable in polynomial time[2] using for example the well known repeated square algorithm. On the other hand, $\texttt{dExp}_{(p,b)}$ is clearly not honest (consider for example $x \in \mathbb{Z}$ such that $|x| \geq 2^{|p|}$), but thanks to proposition 2.29 below, it follows that we can restrict $\texttt{dexp}$ to $\mathbb{Z}_{p-1}$ and therefore the honesty of the function is ensured. In order to invert this function, first of all let's define

$$\texttt{dlog}_{(p,b)}(y) := \{x \in \mathbb{Z} : b^x = y \pmod{\text{p}}\} = \texttt{dexp}^{-1}_{(p,b)}(y)$$

Proposition 2.29. *$\texttt{dlog}_{(p,b)}(y) = [x]_{p-1}$ for a certain $x \in \mathbb{Z}$.*

[2] Remember that the ordinary exponential (with base b) $\texttt{exp}_b : \mathbb{N} \longrightarrow \mathbb{N}$ where $x \mapsto b^x$, is not calculable in polynomial time. Indeed

$$|b^x| = \lfloor x \log_2 b\rfloor + 1 \geq \lfloor 2^{|x|-1}\log_2 b\rfloor + 1 \geq 2^{|x|-1}\log_2 b + 1$$

and an algorithm can't waste more space than time.

Proof. Since $\texttt{dexp}_{(p,b)}$ is surjective then there exists $x \in \texttt{dlog}_{(p,b)}(y)$ and moreover thanks to the Fermat little theroem we have that

$$b^{k(p-1)+x} = (b^{p-1})^k b^x = y \pmod{\text{p}} \quad \forall k \in \mathbb{Z}$$

there fore $[x]_{p-1} \subseteq \texttt{dlog}_{(p,b)}(y)$. Now suppose that $z \in \texttt{dlog}_{(p,b)}(y)$, then $b^{z-x} = 1 \pmod{\text{p}}$. Since $o(b) = p-1$, it follows that $(p-1) \mid (z-x)$ that is $z \in [x]_{p-1}$. □

Remark 2.30. The function $\texttt{dlog}_{(p,b)} : \mathbb{Z}_p^* \longrightarrow \mathbb{Z}_{p-1}$ is a group isomorphism.

The above proposition says that we can find a counter image of $y = \texttt{Dexp}_{(p,b)}$ in the set $\{0, 1, \ldots, p-2\}$. However the brute-force algorithm that performs direct exponentiations modulo p, will have a for cycle that must be iterated at most $p-1$ times, so the worst case analysis of the algorithm says that the running time is exponential in the lenght of the input. Until today there is no a polynomial time (even probabilistic) algorithm that calculates dLog so we have the following assumption:

Conjecture 2.31. (Discrete logarithm assumption) The function $\texttt{dExp}_{(p,b)}$ is a one-way function.

RSA[3] *function*

Consider a natural number $n = pq$ where p and q are two distinct odd prime numbers and pick a number e such that $\gcd(e, \phi(n)) = 1$ (we can suppose that $0 < e < \phi(n)$) where $\phi(n)$ is the Euler totient function. Now define the function

$$\begin{aligned} \texttt{rsa}_{(n,e)} : \mathbb{Z}_n &\longrightarrow \mathbb{Z}_n \\ [x]_n &\longmapsto [x^e]_n \end{aligned}$$

it clearly has a corresponding map between strings that we indicate with $\texttt{RSA}_{(n,e)}$ that operates on the concatenations of the three (bit strings) inputs n, e and x where $0 \leq x \leq n-1$ and outputs the string representation of x^e. This function is honest and calcolable in polynomial time, and as the previous two cases it comes aut that it is extremely hard to invert $\texttt{RSA}_{(n,e)}$. Indeed nowdays it hasn't been found any polynomial time (even probabilistic) algorithm that can invert $\texttt{RSA}_{(n,e)}$ therefore we have

Conjecture 2.32. (RSA assumption) The function $\texttt{RSA}_{(n,e)}$ is a one-way function.

The situation seems to be similar to the other two presented above, but there is a big difference, in fact, roughly speaking, even if it is hard in general to invert $\texttt{RSA}_{(n,e)}$, with the knowledge of a "little" more information we can invert it in polynomial time. This feature is reassumed in the following theorem:

Proposition 2.33. *Fix n and e as above, then there exixts a number d with $0 < d < \phi(n)$ such that $ed = 1 \pmod{\phi(n)}$. Moreover if such element d is given then $\texttt{RSA}_{(n,d)}$ is the inverse function of $\texttt{RSA}_{(n,e)}$.*

Proof. The existence (and the uniqueness) of d is trivial, infact by hypotesis $e \in \mathbb{Z}^*_{\phi(n)}$. We now show that if $[y]_n = \texttt{RSA}_{(n,e)}([x]_n)$, then $y^d = x \pmod{n}$ and this is enough to complete the proof. Since $n = pq$, then $\phi(n) = (p-1)(q-1)$ and it follows that $ed = (p-1)(q-1)k + 1$ for some $k \in \mathbb{Z}$. Moreover

$$y^d = x^{ed} = x^{(p-1)(q-1)k+1} = x \pmod{p}$$

infact if $x = 0 \pmod{p}$, then $x^h = x \pmod{p}$ for every $h \in \mathbb{Z}$, otherwise we apply the Fermat's little theorem. In the same fashion

$$y^d = x^{ed} = x^{(p-1)(q-1)k+1} = x \pmod{q}$$

so by the Chinese remainder theorem we conclude that $y^d = x \pmod{n}$. □

Remark 2.34. Since the exponentiation modulo n can be computated in polynomial time, if one knows such number d then he can invert $\texttt{RSA}_{(n,e)}$ in polynomial time.

[3] RSA is the acronym of the three inventors of this function, namely Rivest, Shamir and Adleman.

One can attack the problem of inverting $\texttt{RSA}_{(n,e)}$ trying to find such an element d. Clearly once that $\phi(n)$ is known then d can be found in polynomial time by solving the diophantine equation $\phi(n)h + ed = 1$, but $\phi(n) = (p-1)(q-1)$ so the factorization of n is needed. We have seen previously that there is no polynomial time algorithm that factorizes n, therefore this strategy to invert $\texttt{RSA}_{(n,e)}$ through $\phi(n)$ fails. From this reasoning it follows easily the following theorem:

Theorem 2.35. *The RSA assumption implies the factoring assumption.*

Proof. If Mult is not one-way then by the above argument $\texttt{RSA}_{(n,e)}$ is not a one-way function too for every couple (n, e). □

Remark 2.36. In other words the RSA assumption is a stronger requirement than the factoring assumption, but the converse of the theorem 2.35 is still an open problem.

Rabin's function

The Rabin's function is very similar to the RSA function, but with essentially an "advantage" respect to it. Let n be a Blum integer i.e. $n = pq$ where p and q are two distinct odd prime numbers such that $p = 3 \pmod 4$ and $q = 3 \pmod 4$ (remember that there are infinite many prime numbers of this form) then define the function

$$\begin{aligned} \texttt{rabin}_n : \mathbb{Z}_n &\longrightarrow \mathbb{Z}_n \\ [x]_n &\longmapsto \left[x^2\right]_n \end{aligned}$$

As usual, $\texttt{RABIN}_n$ is the corresponding function between strings on an input that is the concatenation of n and x represented as bit strings and it is evident that is honest and calculable in polynomial time.

Remark 2.37. Note that $\texttt{RABIN}_n \neq \texttt{RSA}_{(n,2)}$ since $\gcd(\phi(n), 2) \neq 1$.

Proposition 2.38. *Consider a Blum integer $n = pq$ then the following conditions hold for every $y = \texttt{RABIN}_n(x)$ with:*

1) *It has exactly 4 square roots in $\mathbb{Z}_n$ counted with their multiplicity.*
2) *All its square roots can be calculated in polynomial time if the factorization (p, q) is given.*

Proof. 1) By the Chinese remainder theorem we know that $\mathbb{Z}_n \cong \mathbb{Z}_p \times \mathbb{Z}_q$, so $y = x^2 \in \mathbb{Z}_n$ corresponds to an element $(r^2, s^2) \in \mathbb{Z}_p \times \mathbb{Z}_q$. But an element of Q_p, if p is and odd prime, has exactly two square rootsin $\mathbb{Z}_p$, therefore the square roots of y are the 4 elements of $\mathbb{Z}$ corresponding to $(\pm r, \pm s)$.
2) We find explicitly the square roots assuming that $p = 4a + 3$ and $q = 4b + 3$ where $a, b \in \mathbb{N}$. Since p and q are coprime we can find two integers h and k such that $hp + kq = 1$. We now claim that the square roots of y are:

$$\pm u := \pm(hpy^{b+1} + kqy^{a+1}) \pmod n \quad \pm v := \pm(hpy^{a+1} - kqy^{b+1}) \pmod n$$

Once this claim is proved, it is clear that $\pm u$ and $\pm v$ can be calculated in polynomial time if we know the factorization (p, q). We verify only that $u \in \texttt{RABIN}_n^{-1}(y)$ because for the other elements is the same story. Working modulo p we have:

$$u^2 = (hpy^{b+1} + kqy^{a+1})^2 = (kq)^2 y^{2a+2} = (1 - hp)^2 y^{\frac{p+1}{2}} = y^{\frac{p+1}{2}} =$$

$$= y \cdot y^{\frac{p-1}{2}} = y \pmod p$$

where the last equality follows from the well known fract that the group of invertible quadratic residues Q_p^* has order $\frac{p-1}{2}$. In similar way it holds that

$$u^2 = y \pmod q$$

therefore by the Chinese remainder theorem we finally can conclude that $u^2 = y \pmod n$. □

Conjecture 2.39 (Rabin assumption). The function $\texttt{RABIN}_n$ is a one way function for every Blum integer n.

Now we present the most important feature of the Rabin's function that is the "advantage" we have cited earlier:

Theorem 2.40. *For any Blum prime $n = pq$, $\texttt{RABIN}_n$ is a one-way function if and only if* `Mult` *(restricted on Blum integers) is a one-way function.*

Proof. ($\Rightarrow$) Suppose that `Mult` is not one-way, therefore from theorem 2.38 it follows that $\texttt{RABIN}_n$ is not a one-way function.
($\Leftarrow$) First we need a useful preamble: given a number $x^2 \in \mathbb{Z}_n \setminus \{0\}$, we know that there are exactly four square roots of x^2 namely $\pm x$ and $\pm w$ with $x \neq \pm w$ (the 4 roots coincide only when $x^2 = 0$). It follows that

$$0 = x^2 - w^2 = (x + w)(x - w) \pmod{n}$$

where the terms on the right are both nonzero modulo n. In other words we have that $pq \,|\, (x + w)(x - w)$ so $p \,|\, (x + w)$ and $q \,|\, (x - w)$ or viceversa. If for example $p \,|\, (x + w)$ then $p = \gcd(n, x + w)$ and $q = \frac{n}{p}$, so we obtain the factorization of n. We are ready to the real proof: suppose that $\texttt{RABIN}_n$ is not a one-way function, namely there exists a probabilistic algorithm A that inverts in polynomial time $\texttt{RABIN}_n$ with a not negligible probability. The probabilistic algorithm 2 below for factoring n, has polynomial runing time and succeeds with a not negligible probability, so `Mult` is not one-way.

Input: n
$x \leftarrow$ random number in $\mathbb{Z}_n \setminus \{0\}$;
$y \leftarrow x^2$;
$w \leftarrow A(y)$;
if $w \neq \pm x$ **then**
  $s \leftarrow \gcd(x + w, n)$;
  Output: $(s, \frac{n}{s})$
  else
    print: "ERROR";
  end
end

Algorithm 2: factorization of n

□

The point is that if the factoring assumption is true, then we don't need any furter (or stronger) assumption to ensure that $\texttt{RABIN}_n$ is a one-way function. In other words the "advantage" of the Rabin's function is that the problem of its inversion is hard as the problem of factoring that is believed a very difficult problem to solve.
The Rabin's function is a not injective, so in some applications is used a slight modification of it, called the Blum-William function that we know present briefly. Given a Blum integer $n = pq$, then the Blum-Williams function that we indicate as $\texttt{BL-WILL}_n$ is simply $\texttt{RABIN}_n$ restricted to the set of quadratic residues Q_n. The range of $\texttt{BL-WILL}_n$ is evidently Q_n and moreover given an element $y \in Q_n$, using some concepts of number theory it can be shown that exactly one of its four square roots lies in Q_n and it can be easily identified. It follows that the Blum-Williams function is a permutation of Q_n and on the other hand it shares the same computational properties of the Rabin's function.

2.4 Trapdoor permutations and public-key cryptography

We previously said that the one-way functions are the core of the modern cryptography, in fact the encryption function used by Alice should be easy to calculate but hard to invert. However there are the following two problems:

1) If $\mathtt{e}$ is hard to invert for all the world, then Bob will never decrypt the cryptogram in polynomial time. So we need a one-way function $\mathtt{e}$ hiding some "trick" known only to Bob that leads an easy decryption.
2) Our concept of inverting a function is too weak, in fact we said that for inverting f it is enough to find any counter image of $f(x)$ for every x in the domain. On the contrary, if Bob receive the cryptogram $c = \mathtt{e}(m)$ he wants to recover exactly m and not other messages.

The second problem can be overcome simply requiring that $\mathtt{e}$ is a one-way permutation, but on the other hand to solve the first problem we need to define the trapdoor functions namely a particular subclass of the one way functions.

Definition 2.41. A one-way function f is a *trapdoor function* if there exist a string $t_f \in \Sigma^*$ and a TM M such that

- M halts in polynomial time on the input $(f(x), t_f)$ for every x in the domain of f.
- $M(f(x), t_f) \in f^{-1}(f(x))$ for every x in the domain of f.

The string t_f is called a *trapdoor* of the function f.

Definition 2.42.

A *trapdoor permutation* is a one-way permutation that is also a trapdoor function.

Remark 2.43. A trapdoor t_f is the "trick" that permits the easy inversion of f, in fact there is a polynomial time algorithm M that inverts $f(x)$ if we concatenate t_f to the input of M.

It is now evident that the best candidates for being our encryption function are the trapdoor permutations, so we can formally define what is a public key cryptosystem.

Definition 2.44. A public key cryptosystem is a quadruple

$$\big(G, \mathscr{M}, \mathtt{e}(\cdot,\cdot), \mathtt{d}(\cdot,\cdot)\big)$$

where:

- $\mathscr{M} \subseteq \Sigma^*$ is the set of all plaintexts and chipertexts.
- G is a PTM with expected polynomial running time, called the *key generator*, such that $G(1^k) = (i, t_i)$ for every $k \in \mathbb{N}$ where $i \in \Sigma^k$ and $t_i \in \Sigma^*$. The string i is the *public key* (of length k) and t_i is the *private key*.
- For every public key i there is a trapdoor permutation f_i of $\mathscr{M}$ such that t_i is a trapdoor for f_i (namely $t_i = t_{f_i}$).
- For every $m \in \mathscr{M}$ the encryption is given by $\mathtt{e}(m, i) := f_i(m)$.
- For every cypertext $e(m, i) = f_i(m)$, there exists TM M_i that inverts f_i in deterministic polynomial time (knowing the trapdoor), so the decryption is given by $\mathtt{d}(f_i(m), t_i) := M_i(f_i(m), t_i)$.

Remark 2.45. The definition 2.44 is not the standard one, in fact one can relax the requirements allowing for example probabilistic encryptions. For our purposese we don't need to contemplate this case.

To summarize: if Alice and Bob want to communicate using a public key cryptosystem, first Bob generate the two keys i and t_i through G, publishing only the public key i. Then Alice encrypts the messages with the trapdoor permutation f_i (note that the whole world knows the public key i) and Bob can easily decrypt them with the trapdoor t_i and the TM M_i. On the other hand, Eve doesn't know the trapdoor, so, since f_i is in particular a one-way function, she has negligible probability to recover the plaintexts in a reasonable time.
The main weaknesses of public key cryptosystems are basically inherited from the whole structure of modern cryptography: Eve knows how to recover the plaintext, so her knowledge is complete, but she is materially not capable to effectuate the computations. Moreover our definition of one-way function is based on the fact that inverting it on a random input is not feasible, but in reality when Eve tries to intercept a specific communication in a specific context, the sent messages aren't chosen in a completely random way.

2.5 RSA cryptosystem

Below is presented the way in which the RSA cryptosystem works:

- *Key generation:* Bob decides the key length k, then $G(1^k) = ((n,e),d)$ where $n = pq$ with p and q prime numbers, and $e \in \mathbb{Z}^*_{\phi(n)}$. The two primes p and q and are chosen in a random way, $\phi(n)$ is simply $(p-1)(q-1)$, $e \in \mathbb{Z}^*_{\phi(n)}$ is randomly chosen and finally d is calculated in polynomial time with the extended Euclid algorithm. The couple (n,e) is the public key, whereas d is the secret key. Basically G has polynomial expected running time thanks to the density of the prime numbers and since there is a deterministic polynomial time algorithm for testing the primality of a number.
- *Plaintext/Ciphertext set:* We put

$$\mathscr{M} = \{x \in \Sigma^* \,:\, x \text{ is the bit string representation of a number in } \mathbb{Z}_n\}$$

  and if Alice wants to send a longer message, she must break it in peaces such that every peace lies in $\mathscr{M}$.
- *Encryption:* The trapdoor permutation related to the public key (n,e) is $\mathtt{RSA}_{(n,e)}$.
- *Decryption:* The trapdoor corresponding to $\mathtt{RSA}_{(n,e)}$ is the number d and the decryption algorithm consists in calculating the function $\mathtt{RSA}_{(n,d)}$ that was shown to be the inverse of $\mathtt{RSA}_{(n,e)}$. To be more precise we have

$$\mathtt{d}(\mathtt{RSA}_{(n,e)}(x), d) := \mathtt{RSA}_{(n,d)}(\mathtt{RSA}_{(n,e)}(x))$$

Unfortunately for Alice and Bob this not all the story. In fact besides the common problem shared from all public key cryptosystem that we described in the previous section, a "bad usage" of the RSA cryptosystem leads to insecure communications. Our objective is now to explain what we mean with the term "bad usage", presenting some results about the insecurities of the RSA cryptosystem in particularly chosen cases.
First of all it is evident that k should be big as possible, because the incapabilities of Eve to recover the plaintexts come out if she has to manage long inputs (this should be clear also from the role of k in the definition of one-way functions). What about the chose of the two primes p and q? If $p-1$ or $q-1$ is a B-smooth integer, then there is a probabilistic algorithm introduced by Pollard (1972) for factoring $n = pq$ with running time $O(B \cdot \log_2 B \cdot \log_2^2 n)$; so if $B \sim \log_2 n$ we have a probabilistic polynomial time algorithm for breaking the RSA cryptosystem. A similar algorithm was invented by Williams (1982) if $p+1$ or $q+1$ is B-smooth, therefore the security rule is the following: Bob should generate two "safe" primes p and q, namely such that $p \pm 1$ and $q \pm 1$ are B-smooth integers with B large enough.
Another source of insecurity could be the choice of e and consequently of d (remember that for a given e there is a unique d), in fact if d is too small, one can perform the so called Wiener attack based on the following theorem:

Theorem 2.46 (Wiener, 1990). *Suppose that $(n = pq, e)$ is the public key of the RSA cryptosystem with $q < p < 2q$. If $d < \frac{1}{3}n^{\frac{1}{4}}$, then there exists an algorithm that on input (n,e) finds d in polynomial time.*

Proof. [Bon99] □

Finally even with an appropriate choice of p, q and e the two guys should be carefull in encrypting the same message twice, but using different exponents e_1 and e_2:

Proposition 2.47. *Assume that Alice and Bob use the RSA cryptosystem to communicate. If a message m is encrypted twice with two different functions, namely $RSA_{(n,e_1)}$ and $RSA_{(n,e_2)}$ such that $\gcd(e_1,e_2) = 1$ then Eve can recover m in polynomial time.*

Proof. Since e_1 and e_2 are both public and $\gcd(e_1,e_2) = 1$, Eve can compute in polynomial time $h,k \in \mathbb{Z}$ such that $he_1 + ke_2 = 1$. The two ciphertexts are $c_1 = m^{e_1}$ and $c_2 = m^{e_2}$ then working modulo n:

$$m = m^{he_1+ke_2} = c_1^h c_2^k \pmod{n}$$

Practically once that Eve find h and k, she can recover m simply by an exponentiation modulo n. □

There are many other results about the correct usage of RSA cryptosystem in addition to those presented above, but the moral is the following: Alice an Bob, since they're smart guys, have a "warning list" of some things not to do with the RSA cryptosystem, so the communications have an higher level of secrecy.

2.6 Rabin's cryptosystem

The Rabin's cryptosystem is very similar to the RSA cryptosystem:

- *Key generation:* If the key length is k, then $G(1^k) = (n, (p, q))$ where n is a Blum integer and (p, q) is its factorization. The public key is n and the secret key is (p, q).
- *Plaintext/Ciphertext set:* We put

$$\mathscr{M} = \{x \in \Sigma^* \,:\, x \text{ is the bit string representation of a number in } \mathbb{Z}_n\}$$

  as in RSA cryptosystem.
- *Encryption:* We are tempted to say that e is $\mathtt{BL\text{-}WILL}_n$, but note that the domain of this function is Q_n strictly contained in $\mathbb{Z}_n$, so we decide for every message m $e(m, n) = \mathtt{RABIN}_n$. The apparent problem with this definition is that $\mathtt{RABIN}_n$ is a trapdoor function but not a trapdoor permutation since it is not injective, but we can avoid this problem assuming that Alice applies some agreed redundancy to the messages. For example if Alice wants to send the message $m = b_0 b_1 \dots b_r$ to Bob she could send the string $m_t := m b_t b_{t+1} \dots b_r$ where $0 < t < r$.
- *Decryption:* Given the secret key (p, q) we have showed that there exists a polynomial time algorithm that inverts $\mathtt{RABIN}_n$ that we call $\mathtt{RABIN}_n^{-1}$. So Bob performs

$$d(\mathtt{RABIN}_n(m_t), (p, q)) := \mathtt{RABIN}_n^{-1}(\mathtt{RABIN}_n(m_t), (p, q))$$

  and finds four possible plaintexts, but he recognizes the right one thanks to the redundancy. To reduce fatal errors, the redundancy should be long enough, infact squaring a number can lead to a matching of the redundancy.

Remark 2.48. Note that the Rabin's cryptosystem is not perfectly aderent to the definition 2.44 since we have not a trapdoor permutation. However we have used a simpe trick to ensure the uniqueness of the preimage.

Also for the Rabin's cryptosystem one can list some warnings for the encryption. For example k must be big enough and the two primes p and q shoud be choosen in a safe way otherwise the factorization will be easy (see the previous section). These are only few suggestions for a secure communications with the Rabin's cryptosystem, but there are many others: for example the redundancy scheme can be more complicated to avoid attacks that profit by this known feature of the messages.

References

AB09. Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 1 edition, 2009.

Bon99. Dan Boneh. Twenty years of attacks on the RSA cryptosystem. *Notices Amer. Math. Soc.*, 46(2):203–213, 1999.

CLRS09. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition (International Edition)*. The MIT Press, third edition edition, 7 2009.

DH76. W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22, 1976.

GB. Shafi Goldwasser and Mihir Bellare. Lecture notes on cryptography. Online notes.

Gol07. Oded Goldreich. *Foundations of Cryptography: Volume 1, Basic Tools*. Cambridge University Press, 1 edition, 2007.

Gol09. Oded Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, 1 edition, 2009.

HPS08. Jeffrey Hoffstein, Jill Pipher, and J.H. Silverman. *An Introduction to Mathematical Cryptography (Undergraduate Texts in Mathematics)*. Springer, 2008 edition, 2008.

Pap93. Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1 edition, 1993.

Sha49. C. E. Shannon. Communication theory of secrecy systems. *Bell System Tech. J.*, 28:656–715, 1949.

Sip12. Michael Sipser. *Introduction to the Theory of Computation. Michael Sipser*. Thomson South-Western, international ed of 3rd revised ed edition, 2012.

TW06. John Talbot and Dominic Welsh. *Complexity and Cryptography: An Introduction*. Cambridge University Press, 2 2006.