2021-03-02 theory

March 2021

1 Index

1.1 Introduction to Cryptography

Ciphering is the original objective of cryptography. Cryptology (made up of Cryptography and Cryptoanalysis) requires studying mathematics, especially modular arithmetic. Cryptography is the science of using secret codes. A cryptographer is someone who uses and studies secret codes. A cryptanalyst is someone who can hack secret codes and read other people's encrypted messages. Cryptanalysts are also called code breakers or hackers.

The art and science of keeping messages secure is cryptography, and it is practiced by cryptographers. Cryptanalysts are practitioners of cryptanalysis, the art and science of breaking ciphertext; that is, seeing through the disguise. The branch of mathematics encompassing both cryptography and cryptanalysis is cryptology and its practitioners are cryptologists. Modern cryptologists are generally trained in theoretical mathematics they have to be.

Objective of this course is to use python language to solve computer related problems:

1. Communications between computers separated by space.
2. Communications between computers separated by time, e.g., to cipher a hard disk.

In the mid-1970s, electronic communications begin to replace the printed paper in a large number of applications, e.g., communications between many people or many computers in different parts of the world, etc. Therefore, the following problems arise:

- Confidentiality: information and data should be protected from non-authorized readings.
- Authentication: the identity and the origin of information and data should be verified.
- Integrity: information and data should be protected against non-authorized manipulations.

- Non-repudiation: sender of a message should never have the possibility to deny having sent that message.

1.2 Basics

Figure 1.2.1: Encipher & Decipher: symmetric

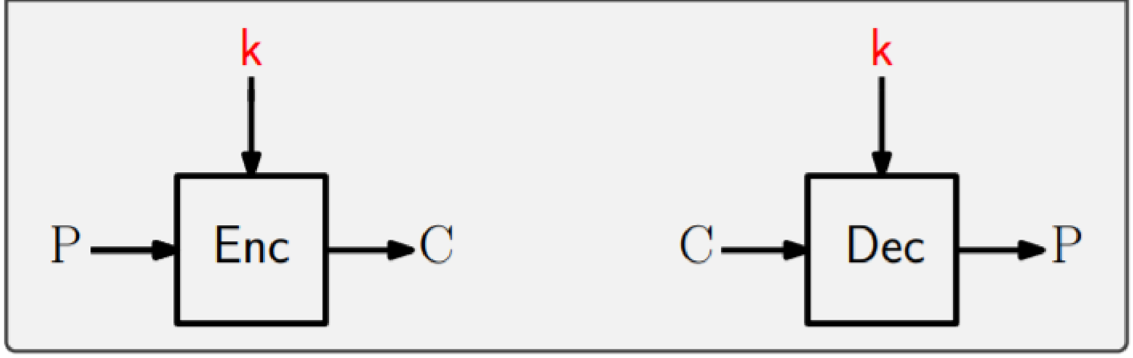
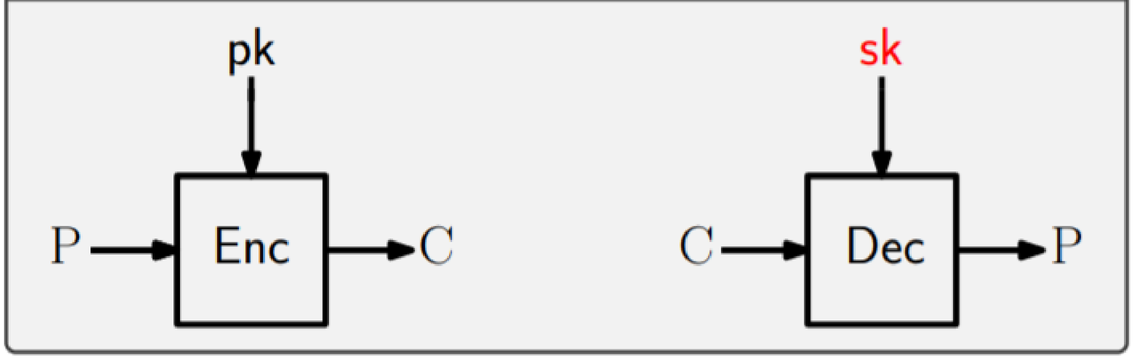


Figure 1.2.2: Encipher & Decipher: Asymmetric



Terminology:

- P = Plaintext, belonging to the $\mathscr{P}$ space (or message $\mathscr{M}$ space
- k = key, belonging to the $\mathscr{K}$ space (brute force attack consists in trying to check all possible keys)
- Enc = encryption algorithm

- C = Ciphertext, belonging to the $\mathscr{C}$ space
- Dec = decryption algorithm
- pk = public key (used in asymmetric cryptography)
- sk = secret key (used in asymmetric cryptography)
- Gen = generator of keys. When it is used to generate the pair (pk,sk) the computation of sk starting from pk should be *computationally unfeasible* (we will call a task computationally infeasible if its cost as measured by eather the amount of memory used or the runtime is finite but impossible large)

A cryptosystem is composed by the two algorithms Enc and Dec (and eventually by a Gen algorithm, which generates the key).
In formulas:

$$\mathbf{E}(\mathrm{k}, \cdot) = \mathbf{E}_{\mathrm{k}} : \mathcal{P} \to \mathcal{C} \qquad\qquad \mathbf{D}(\mathrm{k}, \cdot) = \mathbf{D}_{\mathrm{k}} : \mathcal{C} \to \mathcal{P}$$

$$\forall \mathrm{P} \in \mathcal{P}\ ,\ \mathbf{D}_{\mathrm{k}}(\mathbf{E}_{\mathrm{k}}(\mathrm{P})) = \mathrm{P}$$

To use crypto all parties must agree on all the elements defining the crypto system, that is, the domain parameters of the scheme. In particular, a protocol is an agreement between 2 or more parties.

Kerckhoff's principle
The enemy knows the system (Shannon's Maxim). This means that security should just depend on the secrecy of the key. Another possible definition of this principle could be:
If I take a letter, lock it in a safe, hide the safe somewhere in New York, then tell you to read the letter, that's not security. That's obscurity. On the other hand, if I take a letter and lock it in a safe, and then give you the safe along with the design specifications of the safe and a hundred identical safes with their combinations so that you and the world's best safecrackers can study the locking mechanism and you still can't open the safe and read the letter-that's security.

Brute force attack
To get a clue of the meaning of *computationally infeasible* imagine you are looking for a key k of m bits:

$$k \in \{0,1\}^m$$

```
from timeit import default_timer as timer

#loop with 2**m rounds
m=30
start = timer()
j=0
while j < 2**m:
        # try with key k_j
        print(j)
        j+=1
end = timer()

#print the total time employed to check all keys
print(m, (end - start)/60 ,"minutes") # Time in minutes.
```

Exercise 1.2.5: How long it takes for m=64 bits?

1.3 Three baby examples of symmetric ciphers

Alice have messages constructed by using plaintexts of 5 bits $P \in 20, 1^5$, e.g. P = [11100] and wants to send them to Bob. Oscar is a eavesdropper. From now on, it will considered sufficient a string of 5 bits for teaching purposes, but keys are obviously longer, e.g. 128 bits, 256 bits, etc. In the example, all K, P, C are composed by 5 bits.

1.3.1 The Vernam cipher or $\oplus$-cipher (Gen, Enc, Dec)

The Vernam cipher is, in theory, a perfect cipher. Instead of a single key, each plaintext character is encrypted using its own key. It is also called one-time pad.
To encrypt the message, each character of the plaintext and the key will need to be converted to a numeric code. Fortunately, there are already coding schemes to do this, and we can use standard ASCII codes.
For example, the letter 'H' is 72. This number has a binary representation of 01001000 (using 8 bits).
To apply the Vernam cipher, each bit of the binary character code for each letter of the plaintext is XORed with the corresponding bit of each letter of the binary character code for the corresponding character from the key.
In the below example, the message 'HELLO' will be encrypted using the key 'PLUTO':

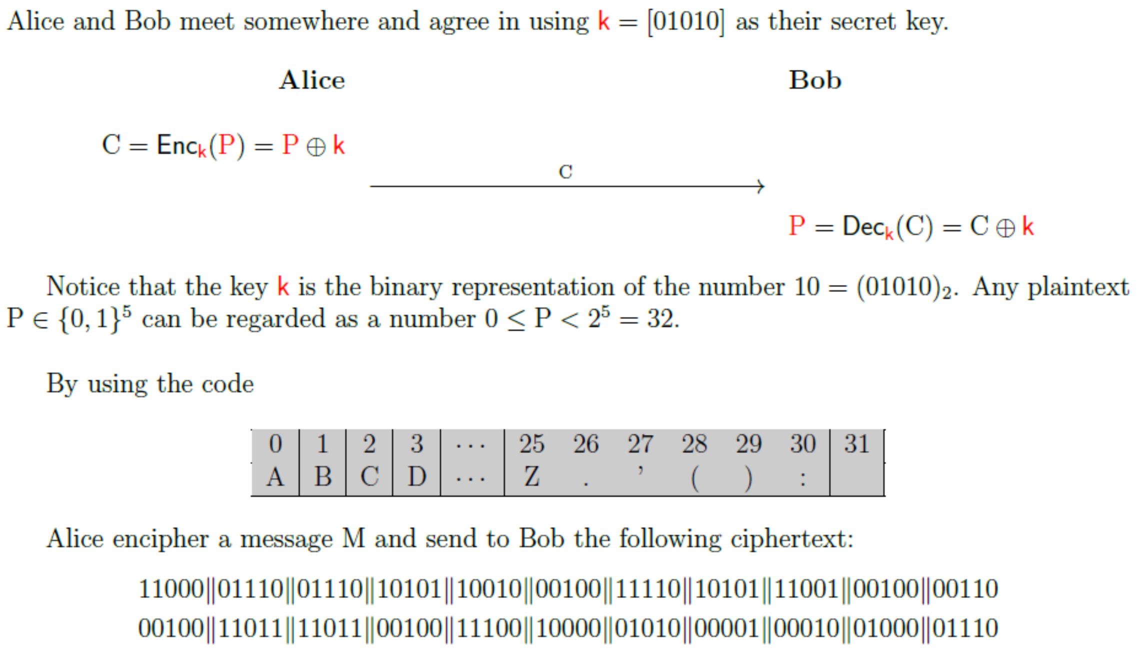
Plaintext
H - 01001000
E - 01010000
L - 01001100
L - 01001100
O - 01001111
Key

P - 01010000
L - 01001100
U - 01010101
T - 01010100
O - 01001111

Ciphertext

00011000
00011100
00011001
00011000
00000000

Antother example: Alice and Bob meet somewhere and agree in using k = [01010] as their secret key.



Alice and Bob meet somewhere and agree in using k = [01010] as their secret key.

Alice **Bob**

$C = \mathsf{Enc}_k(P) = P \oplus k$

$\xrightarrow{C}$

$P = \mathsf{Dec}_k(C) = C \oplus k$

Notice that the key k is the binary representation of the number $10 = (01010)_2$. Any plaintext $P \in \{0,1\}^5$ can be regarded as a number $0 \leq P < 2^5 = 32$.

By using the code

0	1	2	3	...	25	26	27	28	29	30	31
A	B	C	D	...	Z	.	'	(	)	:	

Alice encipher a message M and send to Bob the following ciphertext:

11000‖01110‖01110‖10101‖10010‖00100‖11110‖10101‖11001‖00100‖00110
00100‖11011‖11011‖00100‖11100‖10000‖01010‖00001‖00010‖01000‖01110

The one used above is a code dictionary, which connects objects to string of bits. The most famous code dictionary is the ASCII code.

Example: Having a plaintext character P = [11011] and the key chosen by Alice and Bob, the Vernam cipher will return P ⊕ k = [11011] ⊕ [01010] = [10001].

Exercise 1.3.2: Having as input M and C we can extract the key of each part of the encryption. For example, if M_0 = [00100] and C_0 = [10010], we can notice that, in order to obtain C starting from M, the current key K should be $K_0 = [10110]$.

1.3.2 The Caesar cipher or ⊞-cipher (Gen, Enc, Dec)

Here instead the $\oplus$ we use the $\boxplus$ operation which means the sum with the carry bit up to 5 bits.
For example:

$$11011 \boxplus 10101 = 10000$$

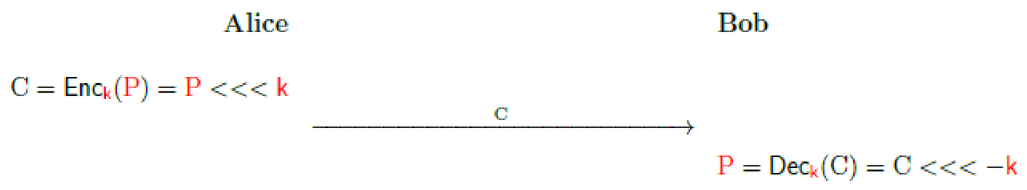
The $\boxplus$ operation is easy to see by using base 10 representation and arithmetic modulo $2^5 = 32$:

1.3.3 The rot cipher or circular shift <<< r (Gen, Enc, Dec)

In this cipher instead of $\oplus$ or $\boxplus$ we use the operation $<<<$ r. For example:

$$[110111] <<< 3 = [111110]$$

Namely the bits are shifted toward left in a circular way.



For this cipher the key k used by Alice and Bob is an integer.

Exercise 1.3.3: The rotation can be made at most of n bits, since the shift operation is circular. So, given a ciphertext, we can easily find out the used key by trying all the possible shifts operations, until we get a readable message. For example, if key length is n = 5 bits, we have to try only 5 different rotations.

1.4 Basics of Modular Arithmetic

Modular Arithmetic deals with the study of the finite modular ring, composed by the remainders of division by n:

$$\mathbb{Z} = \mathbb{Z}/n \cdot \mathbb{Z} = \{0,1,\cdots,(n\text{-}1)\}$$

For example:

$$\mathbb{Z}_2 = \{0,1\}$$
$$\mathbb{Z}_3 = \{0,1,2\}$$

To get the remainder $r \in \mathbb{Z}_n$ of the integer $a \in \mathbb{Z}$ modulo n in python: r = a%n.
In mathematics such operation is written as

$$r = a \pmod{n}$$

whose meaning is that

$$a = n \cdot q + r$$

where r belongs to {0,1,$\cdots$,(n-1} and $q \in \mathbb{Z}$.
A ring is a set with 3 operations: $+$, -, $\times$.
Example:
In $\mathbb{Z}_6$ these operations below are possible:

$$4 + 4 = 8 mod 6 = 2$$
$$2 - 5 = -3 mod 6 = 3$$
$$3 \times 4 = 12 mod 6 = 0$$

Particular case:
If $a \times b \equiv 0(modp)$, then $a \equiv 0(modp)\mathbf{or}b \equiv 0(modp)$.
In fact, if we assume that, for example, $a \equiv 0(modp)$, we obtain:

$$b \equiv \frac{a \times b}{a} \equiv \frac{0}{a} \equiv 0(modp)$$

Note that, if p is a **prime number** the ring $\mathbb{Z}_p$ is also denoted as $\mathbb{F}_p$ or GF(p). Actually, means Galois Field. A field is a ring but with 4 operations: $+$, -, $\times$, $/$.

Moreover, $\mathbb{Z}_3 = \{0, 1, 2\}$ Field can be used to compute the *classical* inverse of a number when performing a division by a number (by a remainder). For example:

$$2^{-1} = \frac{1}{\mathbf{2}} = 2(mod3)$$

because 2 $\times$ **2** = 1 (mod 3).

$\mathbb{Z}_3$ provides *standard* inverse, but it's not valid only for $\mathbb{Z}_3$: every other ring provides its own inverse:

$$\frac{\spadesuit}{\diamondsuit} = \spadesuit \cdot \frac{1}{\diamondsuit}$$

For example:
In $\mathbb{Z}_{11}$:

$$\frac{1}{9} \equiv 9^-1 \equiv 5(mod 11)$$

While in $\mathbb{Z}_7$:

$$\frac{3}{5} \equiv 3 \cdot \frac{1}{5} \equiv 3 \cdot 3 \equiv 9 \equiv 2(mod 7)$$

Exercise 1.4.2: Show that $a \equiv b(mod n)$ if and only if n divides $a - b$:

- Step 1: Assume $a\ mod\ n = b\ mod\ n$ and prove n divides $a - b$.
  Since $a\ mod\ n = b\ mod\ n$, we can write $a = q1n + r$ and $b = q2n + r$. Then $a - b = (q1 - q2)n$ is divisible by n.

- Step 2: Assume n divides $a - b$ and prove $a\ mod\ n = b\ mod\ n$.
  We know we can write $a = q1n + r1$ and $b = q2n + r2$, with remainders $r1$ and $r2$ both between 0 and n.
  Then $a - b = (q1 - q2)n + (r1 - r2)$. Because n goes evenly into $(q1 - q2)n$, the remainder when $a - b$ is divided by n is the same as the remainder when $r1 - r2$ is divided by n.
  Since $a - b$ is divisible by n, the remainder when $r1 - r2$ is divided by n must be 0. So $r1 - r2$ is a multiple of n.
  But $r1$ and $r2$ are both numbers between 0 and n, so the only way $r1 - r2$ can be an even multiple of n is for it to be equal to $0 \cdot n = 0$.
  So $r1 = r2$ and $a\ mod\ n = b\ mod\ n$.