Cryptography

A (nearly) complete overview

Gaspare Ferraro

April 1, 2019

Visit us!
zenhack.it

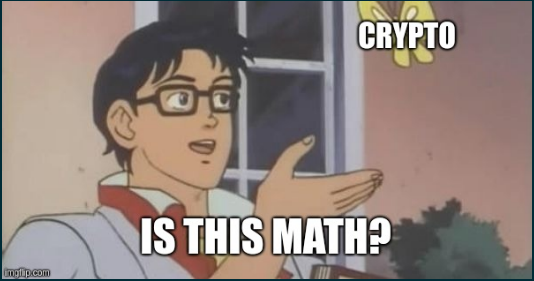
@GaspareG
ferraro@gaspa.re

Table of Contents

Table of Contents

Warning!

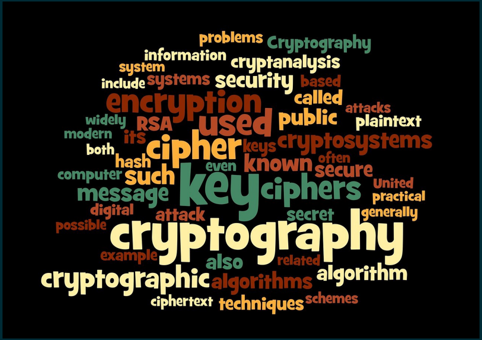
In this lesson we will use *math*!



It wasn't always like that though ...

Why cryptography?

Cryptography (from greek: kryptos "hidden, secret" and graphein, "to write")



The science of secure communication

Cryptography yesterday

(a) Caesar Cipher

(b) Scitala

Cryptography today

The needs, as well as the available resources, have evolved and today we can divide cryptography into:

(EN|DE)CRYPTION	**ASYMMETRIC (RSA, ECC, ...)** **SYMMETRIC (DES, AES, ...)**
KEY EXCHANGE	**RSA, DH, ECDH, ...**
AUTHENTICATION	**RSA, DSA, ECDSA, ...**
HASHING	**MD5, SHA-1, SHA-256, ...**

Table of Contents

What is a message?

A message is a sequence of symbols used to communicate

A symbol of the message is called character

The set of all the possibile characters is called alphabet

The set of all the possible (meaningful) messages is called dictionary

ASCII encoding

ASCII = American Standard Code for Information Interchange

char encoded in 7 bits + 1 bit for check (parity bit).

ASCII (1977/1986)

	_0	_1	_2	_3	_4	_5	_6	_7	_8	_9	_A	_B	_C	_D	_E	_F
0_ 0	NUL 0000	SOH 0001	STX 0002	ETX 0003	EOT 0004	ENQ 0005	ACK 0006	BEL 0007	BS 0008	HT 0009	LF 000A	VT 000B	FF 000C	CR 000D	SO 000E	SI 000F
1_ 16	DLE 0010	DC1 0011	DC2 0012	DC3 0013	DC4 0014	NAK 0015	SYN 0016	ETB 0017	CAN 0018	EM 0019	SUB 001A	ESC 001B	FS 001C	GS 001D	RS 001E	US 001F
2_ 32	SP 0020	! 0021	" 0022	# 0023	$ 0024	% 0025	& 0026	' 0027	(0028	) 0029	* 002A	+ 002B	, 002C	- 002D	. 002E	/ 002F
3_ 48	0 0030	1 0031	2 0032	3 0033	4 0034	5 0035	6 0036	7 0037	8 0038	9 0039	: 003A	; 003B	< 003C	= 003D	> 003E	? 003F
4_ 64	@ 0040	A 0041	B 0042	C 0043	D 0044	E 0045	F 0046	G 0047	H 0048	I 0049	J 004A	K 004B	L 004C	M 004D	N 004E	O 004F
5_ 80	P 0050	Q 0051	R 0052	S 0053	T 0054	U 0055	V 0056	W 0057	X 0058	Y 0059	Z 005A	[005B	\ 005C	] 005D	^ 005E	_ 005F
6_ 96	` 0060	a 0061	b 0062	c 0063	d 0064	e 0065	f 0066	g 0067	h 0068	i 0069	j 006A	k 006B	l 006C	m 006D	n 006E	o 006F
7_ 112	p 0070	q 0071	r 0072	s 0073	t 0074	u 0075	v 0076	w 0077	x 0078	y 0079	z 007A	{ 007B	| 007C	} 007D	~ 007E	DEL 007F

Letter Number Punctuation Symbol Other undefined Changed from 1963 version

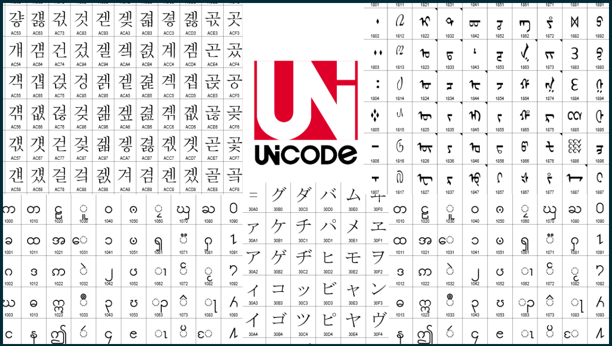
$0, \ldots, 31 + 127 \rightarrow$ non-printable chars (null, new line, tab, others)
$32, \ldots, 126 \rightarrow$ printable chars (letters, digits, punctuation, others)

Extended ASCII $\rightarrow$ char encoded in 8 bits (add 128 printable chars to standard ASCII)

Unicode encoding

Obviously 128 (or 256) characters are not enough!
(Chinese, cyrillic, greek alphabets, emojis...)

Different standards: UTF-8, UTF-16, UTF-32 and others

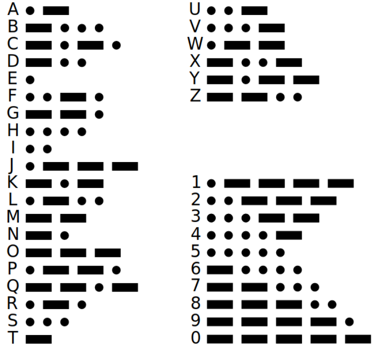


Currently assigned "only" 137993 characters

Morse code

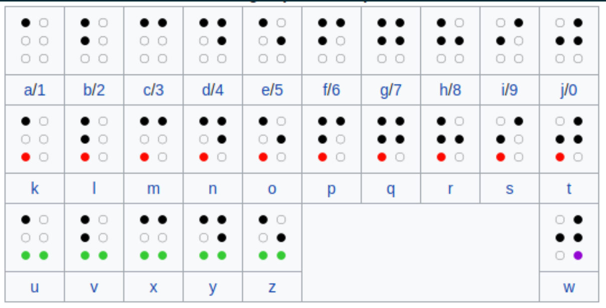
(Audio) character encoding scheme used in (telegraph) telecommunication.

Each character is encoded using a combination of short/long signal.



Braille code

(Tactile) character encoding scheme used for visually impaired people.
Each characted is encoded using a 2×3 rectangle with ”raised dots”.



Base64

Group message in blocks of 6 bits.

Advantage: encode all the ASCII chars in printable characters

<table>
<tr><th>source ASCII (if <128)</th><th colspan="8">M</th><th colspan="8">a</th><th colspan="8">n</th></tr>
<tr><th>source octets</th><td colspan="8">77 (0x4d)</td><td colspan="8">97 (0x61)</td><td colspan="8">110 (0x6e)</td></tr>
<tr><th>Bit pattern</th><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td></tr>
<tr><th>Index</th><td colspan="6">19</td><td colspan="6">22</td><td colspan="6">5</td><td colspan="6">46</td></tr>
<tr><th>Base64-encoded</th><td colspan="6">T</td><td colspan="6">W</td><td colspan="6">F</td><td colspan="6">u</td></tr>
<tr><th>encoded octets</th><td colspan="6">84 (0x54)</td><td colspan="6">87 (0x57)</td><td colspan="6">70 (0x46)</td><td colspan="6">117 (0x75)</td></tr>
</table>

Valore	ASCII	Valore	ASCII	Valore	ASCII	Valore	ASCII
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/

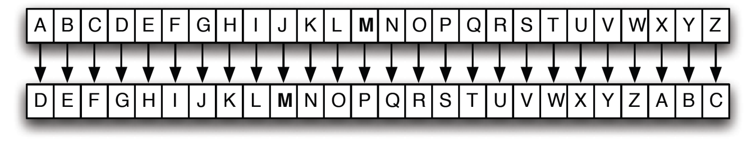
Message are padded with = (e.g. *flag* → *ZmxhZwo=*)

Table of Contents

Caesar cipher

Encrypt: right shift each letter of 3 positions

Decrypt: left shift each letter of 3 positions

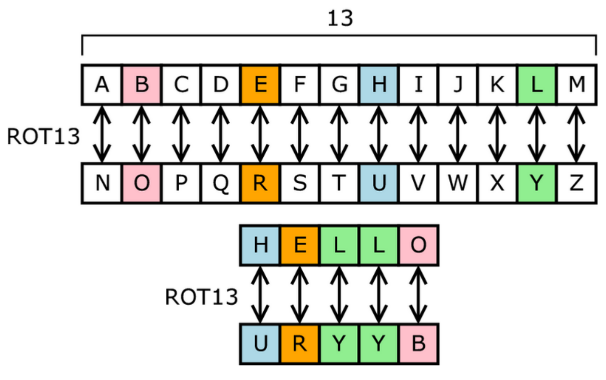


General cipher: shift letter of K positions

Attack: bruteforce all the possible K (only 25 values...)

ROT{13, 47}

ROT13: Caesar cipher with $K = 13$ on alphabetic dictionary
ROT47: Caesar cipher with $K = 47$ on printable ASCII chars (33 - 126).



Why $K = 13$ (or $K = 47$)? Because Encrypt = Decrypt

Classical ciphers

Substitution ciphers

- Monoalphabetic ciphers: $C_{new} = P[C_{old}]$ (Where P is a dictionary permutation)

(ROT-K is a monoalphabetic cipher with P is a cyclic rotation)

- Polialphabetic ciphers: multiple substitution alphabets

(more than one dictionary permutation)

Transposition ciphers

Encryption systems where the positions held by units of plaintext (characters or groups of characters) are shifted according to a regular system.

E.g. We want to encrypt the message *WE ARE DISCOVERED. FLEE AT ONCE* using the **route cipher**:

Grid:

```
W R I O R F E O E
E E S V E L A N J
A D C E D E T C X
```

Cipher text: *EJXCTEDECDAEWRIORFEONALEVSE*

Substituion cipher: XXX

Transposition cipher: XXX

dcode.fr

https://www.dcode.fr/tools-list

Almost all possible classic ciphers (old and new), encoder/decoder, ...

Cryptanalysis

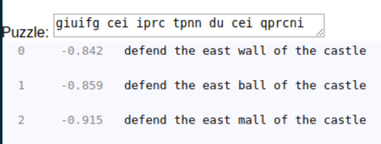
Often the vulnerability is not in the algorithm but in its application...

- Bad use of the key (too short, reused, bad generated, ...)
- Messages use a poorly distributed dictionary
- We know the message format (e.g.: $\mathrm{FLAG}\{\dots\}$)

In particular we talk about statistical cryptanalysis when we force the cipher not from algorithmic point of view but from statistical one

For example in english the character E has a frequency of 12.02% while Z only 0.07%

Useful tool (for substitution ciphers): https://quipqiup.com



Attack models

Classification of cryptographic attacks:

- Ciphertext-only attack: access only to the ciphertext, and has no access to the plaintext
- Known-plaintext attack: access to at least a limited number of pairs of plaintext and the corresponding enciphered text
- Chosen plaintext attack: able to choose a number of plaintexts to be enciphered and have access to the resulting ciphertext (encrypt oracle)
- Chosen ciphertext attack: able to choose arbitrary ciphertext and have access to plaintext decrypted from it (decrypt oracle)
- Side-channel attack: use of other informations to break the cipher (time, sound, power, error, ...).

Table of Contents

Symmetric-key cryptography

Symmetric ciphers are those where messages are encrypted and decrypted using the same key, which must be known only and exclusively to the two parts

$$\mathcal{C}(m, k) = c \text{ (encrypt function)}$$
$$\mathcal{D}(c, k) = m \text{ (decrypt function)}$$

Obviously:

$$\mathcal{D}(\mathcal{C}(m, k), k) = m$$

The original message is not altered during the communication

E.g. In the caesar cipher:

$$\mathcal{C}(m, k) = \text{right shift of } k \text{ positions each character}$$
$$\mathcal{D}(c, k) = \text{left shift of } k \text{ positions each character}$$

Shannon principle

How to assess whether a cipher is robust enough?
(Where robustness means its probability of being successfully attacked)

Shannon defines two key concepts:

- Confusion: the key must be well distributed in the encrypted message (each bit of the cipher should depend on each bit of the key with probability 50%).
- Diffusion: the message must be well distributed in the encrypted message (each bit of the cipher should depend on each bit of the message with probability 50%).

In the Caesar cipher we have no kind of diffusion and low confusion (why?)

XOR cipher

Consider the XOR (exclusive or) operation $\oplus$, the following properties are valid:

- $0 \oplus 0 = 1 \oplus 1 = 0$
- $0 \oplus 1 = 1 \oplus 0 = 1$
- $x \oplus y \oplus y = x$

We define the XOR cipher as:

$\mathcal{C}(m,k) = m \oplus k \qquad (m[i] \oplus k[i],\ 0 \leq i < ||m||)$

$\mathcal{D}(c,k) = c \oplus k \qquad (c[i] \oplus k[i],\ 0 \leq i < ||c||)$

Problem: the key k could be shorter than the message m.

Solution: reuse the key as $k' = k \cdot k \cdot \ldots \cdot k$ until $||k'||$ m.

Example:

`m = 01100011 01101001 01100001 01101111` (ciao in ASCII).

`k = 01111000 01111000 01111000 01111000` (x in ascii 4 times)

`c = 00011011 00010001 00011001 00010111` (non printable, `GxEZFw==` in b64)

One-time pad

The problem with the XOR cipher is that encrypting repeatedly reusing the same key can leak statistical informations of the original message

We call Vernam cipher (or one-time pad) a XOR cipher where the key has the same length of the message.

This cipher is called perfect because we have that:

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

The probability that M is a certain message m knowing that the cipher C is c is equal to the probability that M is a certain message not knowing the cipher (all messages are equiprobable, the encrypted message does not give us any information about the real message).

Nice in theory, but:

- The key must be exchanged using a secure method (exchange them by *hand*).
- The key must be generated randomly and not used (otherwise a many-time pad attack is possible).

Many-time pad & XorTool

Nice article: thecrowned.org/the-one-time-pad-and-the-many-time-pad-vulnerability

XorTool: tool for statistical analysis of encrypted messages:

```
root@ddos:~/Desktop/xortool/xortool# xortool binary_xored
The most probable key lengths:
   1:   9.6%
   5:   15.0%
  10:   21.7%
  15:   9.3%
  20:   13.6%
  25:   6.0%
  30:   9.1%
  35:   4.2%
  40:   6.6%
  50:   5.0%
Key-length can be 5*n
Most possible char is needed to guess the key!
```

Knowing the initial part, we can see words in the message:

```
This is clas**********l**
Do not share**********}
{FLG:ch3ck_e*****
```

Going by trial the final flag is reconstructed:

```
This is classified*********
Do not share the s*****
{FLG:ch3ck_em@il}
```

Block vs Stream ciphers

Block ciphers

- Works with fixed-length groups of bits (called blocks)
- More memory/time requirements
- High diffusion and confusion
- Error propagation
- Need to handle messages length (padding)

Famous ciphers:

DES
AES
BlowFish

Stream ciphers

- Works by encrypt digits one at the time
- Faster encryption/decryption
- Low diffusion
- Low propagation error
- Need a key stream (usually a shift register)

Famous ciphers:

ChaCha
Salsa20
LFSR-based

Padding a message (PKCS#5 & PKCS#7)

How to handle messages of length not multiple of the block size?

Idea: append "some chars" to the message (padding string)

PKCS#5:
The padding string PS shall consist of $8 - (||M|| \mod 8)$ *octets all having value* $8 - (||M|| \mod 8)$.

PKCS#7:
For such algorithms, the method shall be to pad the input at the trailing end with $k - (l \mod k)$ *octets all having value* $k - (l \mod k)$*, where* l *is the length of the input.*

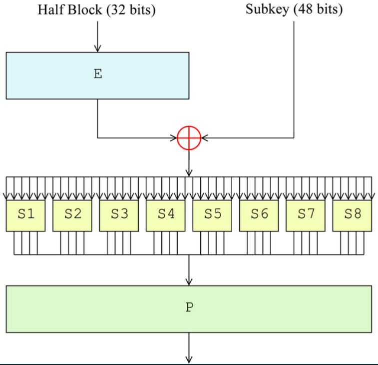
Why $8 - (||M|| \mod 8)$ and not $(||M|| \mod 8)$?

DES

Data Encryption Standard

Developed in 1975 by Feistel, encrypt blocks of 64 bits with a 56 bits key

Implements the confusion and diffusion principle by 16 rounds of the Feistel function



The Feistel function consists in 4 stages:

- 1. Expand the half block from 32 to 48 bits (E-Box)
- 2. Mix result and subkey using a XOR operation
- 3. Substitution of the 6-bits input with a 4-bits output according to a lookup table (S-Box)
- 4. Permutation of the result (P-Box)

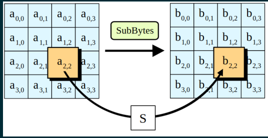
Problem: DES vulnerable to a bruteforce attack (only 56 bits for the key...)

AES

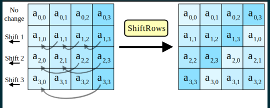
Advanced Encryption Standard

AES replace DES starting from 2001 and is currently the standard in secure communications (TLS1.3 support only AES and ChaCha20)

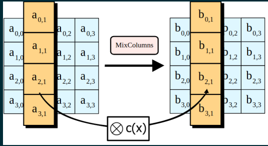
Based on a substitution–permutation network (equivalent of the Feistel network)



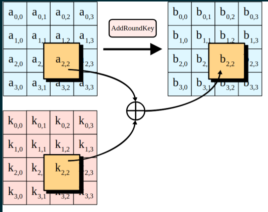
Step 1:
Each byte is replaced with another according to a lookup table (S-Box)



Step 2:
Transposition of each rows by 0, 1, 2 or 3 positions



Step 3:
Linear mixing operation where each column is mapped into a new one



Step 4:
Each byte is XORed with the corresponding value of the subkey

Block cipher mode of operation

How to ciphers two or more blocks? Different modes, different features:

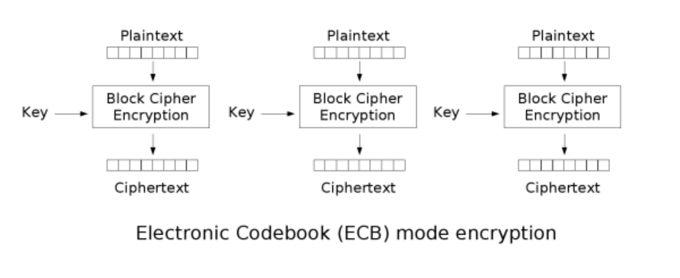
- Parallel encryption: encrypt different blocks at the same time (without race conditions)
- Parallel decryption: decrypt different blocks at the same time (without race conditions)
- Random read: decrypt any single block without decrypting the previous ones

Mode	Parallel encryption	Paralllel decryption	Random read
Electronic Code Book (ECB)	Yes	Yes	Yes
Cipher Block Chaining (CBC)	No	Yes	Yes
Propagating CBC (PCBC)	No	No	No
Cipher Feedback (CFB)	No	Yes	Yes
Output Feedback (OFB)	No	No	No
Counter (CTR)	Yes	Yes	Yes

ECB (Electronic Code Book)

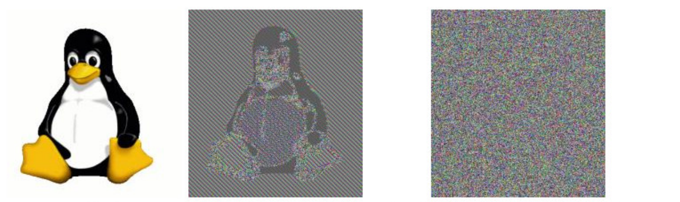
The message is divided into blocks, and each block is encrypted/decrypted separately:

$$C_i = f(M_i, Key)$$
$$M_i = f(C_i, Key)$$



Problem: no diffusion, ECB encrypt same plaintext in same ciphertext

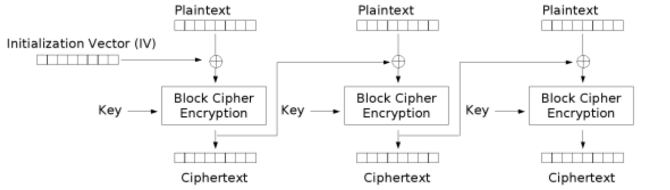
How to break ECB (padding-oracle attack)



Original *Encrypted using ECB mode* *Modes other than ECB result in pseudo-randomness*

CBC (Cipher Block Chaining)

In CBC each block of plaintext is XORed with the previous ciphertext block before being encrypted. An initialization vector in needed for the first block.



Cipher Block Chaining (CBC) mode encryption

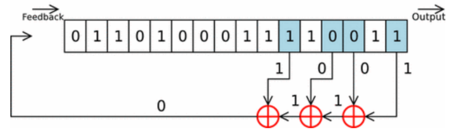
$$C_0 = IV$$
$$C_{i+1} = \mathcal{C}(M_i \oplus C_i, Key)$$
$$M_{i+1} = \mathcal{D}(C_{i+1}, Key) \oplus C_i$$

Stream cipher: LFSR

Linear-Feedback Shift Registers

Shift register whose input bit is a linear function of its previous state

```
class LFSR:
  def __init__(self, register, branches):
    self.register = register
    self.branches = branches
    self.n = len(register)
  def next_bit(self):
     ret = self.register[self.n - 1]
     new = 0
     for i in self.branches:
       new ^= self.register[i - 1]
     self.register = [new] + self.register[:-1]
     return ret
```




```
register = [0,1,1,0,1,0,0,0,1,1,1,1,0,0,1,1]
branches = [10,12,13,15]
gen = LFSR(register, branches)
for c in stream:
  print(c ^ gen.next_bit())
```

Table of Contents

Public-key cryptography

Modular arithmetic

RSA

An example...

How to choose parameters

Break RSA (online approach)

Break RSA (offline approach)

Table of Contents

Key exchange

Diffie-Hellman key exchange

A man-in-the-middle attack to DH

Table of Contents

Hash function

Why hash

How to store passwords

How to store passwords[2]

How to (not) store passwords

Proof of Work

MD5

SHA{0, 1, 2}

Finding collision

Reverse an hash function (online approach)

Reverse an hash function (offline approach)

fcrackzip

JohnTheRipper

Hashcat

Table of Contents

Steganography

File stego (file, binwalk, exiftool, strings)

Image stego

Layer stego

Audio stego

Morse code

Spectography analysis (Audacity)

AudioStego

DeepSound