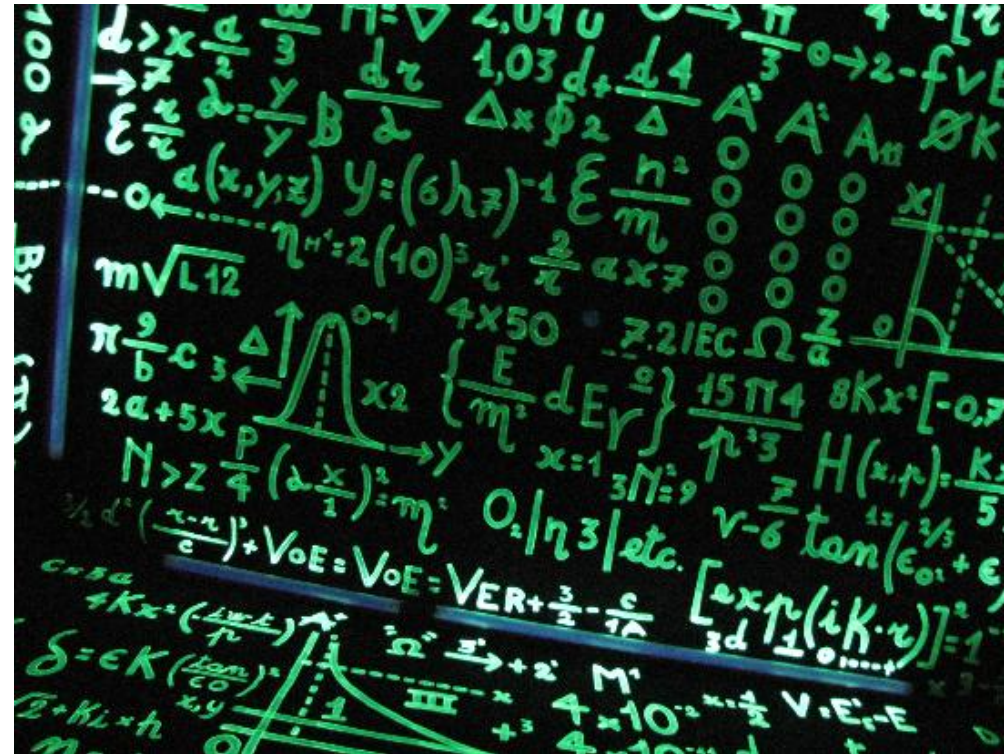


# Laboratorio di Crittografia



## Il Pensiero Computazionale

Percorso Formativo per i Docenti della Scuola Secondaria di II Grado

## Lezione 3

Stefano Forti and Davide Neri

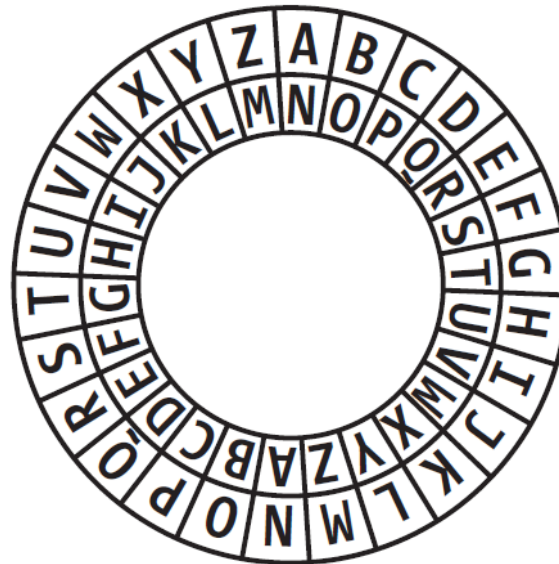
# Cosa faremo oggi ?

- [Introduzione al Python 🐍]
- Cifrario di Cesare
- One Time Pad
- Diffie-Hellman e l'Esponenziazione Veloce

# Cifrario di Cesare (Idea)

Un *cifrario* è un codice segreto che trasforma un messaggio in modo da renderlo incomprensibile a chi non conosce la chiave del codice.

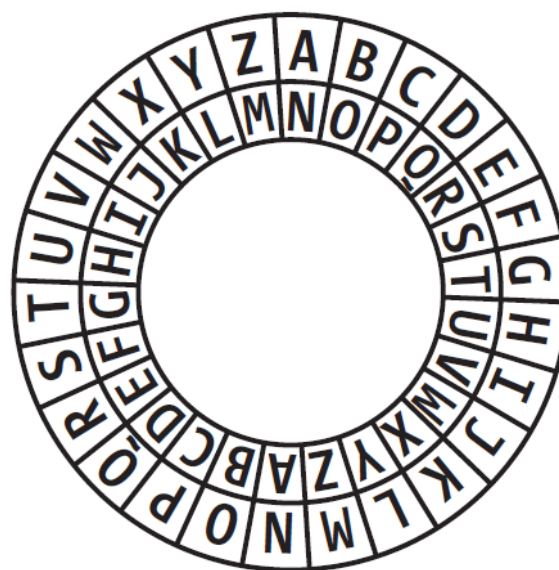
Giulio Cesare (100-44 a.C.) inviava messaggi facendo "scorrere" l'alfabeto come mostrato dall'anello qui sotto:



La *chiave* del cifrario (o *rotazione*) è rappresentata dal numero di posizioni  $k$  di cui far scorrere ciascuna lettera del messaggio ( $k = 13$  nell'esempio).

# Cifrario Simmetrico

Scegliendo  $k = 13$  come chiave, otteniamo che  $l \rightarrow l'$  e che  $l' \rightarrow l$  dove  $l, l'$  rappresentano lettere dell'alfabeto.



Un cifrario di questo tipo si dice *simmetrico* in quanto permette di **crittare** e **decrittare** un messaggio eseguendo la stessa operazione.

Per esempio: HELLO  $\rightarrow$  URYYB  $\rightarrow$  HELLO .

# Due Utili Funzioni

Per il laboratorio di oggi, iniziamo col definire due funzioni molto utili hanno bisogno di

```
import string
```

La prima funzione, dato un numero `n` tra `0` e `25`, ci restituisce il carattere dell'alfabeto inglese in quella posizione:

```
def from_value_to_char(n):  
    alphabet = list(string.ascii_uppercase)  
    return alphabet[n]
```

La seconda, dato un carattere `c`, ci restituisce la sua posizione nell'alfabeto inglese:

```
#TODO: provate a scriverla voi...  
def from_char_to_value(c):
```

# Due Utili Funzioni

Per il laboratorio di oggi, iniziamo col definire due funzioni molto utili hanno bisogno di

```
import string
```

La prima funzione, dato un numero `n` tra `0` e `25`, ci restituisce il carattere dell'alfabeto inglese in quella posizione:

```
def from_value_to_char(n):  
    alphabet = list(string.ascii_uppercase)  
    return alphabet[n]
```

La seconda, dato un carattere `c`, ci restituisce invece la sua posizione nell'alfabeto inglese:

```
def from_char_to_value(c):  
    alphabet = list(string.ascii_uppercase)  
    return alphabet.index(c)
```

# Cifrario di Cesare (Codice)

```
def cifrario_di_cesare(msg):  
    msg = msg.upper()  
    output = ""  
  
    for l in msg:  
  
        #TODO: completare il codice in modo da gestire anche [' ', '.', ',', '!', '?', ':']  
  
        output += l  
    return output
```

# Cifrario di Cesare (Codice)

```
def cifrario_di_cesare(msg):  
    msg = msg.upper()  
    output = ""  
  
    for l in msg:  
        if not(l in [' ', '.', ',', '!', '?', ':']):  
            v = (from_char_to_value(l) + 13) % 26  
            l = from_value_to_char(v)  
        output += l  
    return output  
  
print(cifrario_di_cesare('URYB, URYB!'))
```



# Prova Tu!

A chi è destinato il messaggio che Eva ed Eustachio hanno rubato a una famosa spia internazionale?

**FVYIVN, EVZRZOEV NAPBEN?**



# Esercizio 1

Definire la funzione `def cifra_cesare(m, k):` e la funzione `def decifra_cesare(m, k):` che permettano l'utilizzo di una chiave `k` arbitraria per cifrare e decifrare il messaggio in ingresso `m`.

# One Time Pad (OTP)

Si tratta del cosiddetto *cifrario perfetto* che fu utilizzato per le comunicazioni a partire dalla Seconda Guerra Mondiale dalle spie americane e sovietiche.

L'uso più celebre rimane quello di cifratura delle comunicazioni sulla *Linea Rossa* tra il Cremlino e la Casa Bianca negli anni successivi alla crisi dei missili di Cuba (1963).

LFHHY ZAHBB JRNXX BYMFW KQZAT	A ABCDEFGHIJKLMNOPQRSTUVWXYZ
VRETH JPCBU RUSTQ JXKXN ELDEL	ZYXWVUTSRQPONMLKJIHGFEDCBA
PODYF JVLUV XFEKL HPLGA ZXYZY	B ABCDEFGHIJKLMNOPQRSTUVWXYZ
TSUIO XBKKI NBEND HPNPI OZVOZ	YXWVUTSRQPONMLKJIHGFEDCBA
EYJFW DBKKR PNTVY YTKKK ATOPR	C ABCDEFGHIJKLMNOPQRSTUVWXYZ
NHCJK FPNBV BRZZN QQZYN CYSDS	XWVUTSRQPONMLKJIHGFEDCBA
YIIUJ TWRRE QHRDE YOVRJ HOCBY	D ABCDEFGHIJKLMNOPQRSTUVWXYZ
HALOK NHIIN CAIDV RDTKH ZDZHP	WVUTSRQPONMLKJIHGFEDCBA
GINDS CHOFV KBBVJ CAYSO IABHU	E ABCDEFGHIJKLMNOPQRSTUVWXYZ
KZZK OZJIM DBRCY BNDVZ LFBKT	VUTSRQPONMLKJIHGFEDCBA
TI WIFN INNSF RUVCV UITRN	F ABCDEFGHIJKLMNOPQRSTUVWXYZ
NQONG ZUBZB EPVJI MCZXY FSTEX	UTSRQPONMLKJIHGFEDCBA
VEIOE HDVTN GSSNG LRZFG UKUGK	G ABCDEFGHIJKLMNOPQRSTUVWXYZ
POPRI GCFAA NLTKE DANDQ QAIHU	TSRQPONMLKJIHGFEDCBA
HEINQ LBTVP HVBXN HNUUK ACPXA	H ABCDEFGHIJKLMNOPQRSTUVWXYZ
AYGFS ZNPOU SYNXX IYIPD RJCEK	SRQPONMLKJIHGFEDCBA
PRPDP JFWIO NYLIX GTHC QKXKH	I ABCDEFGHIJKLMNOPQRSTUVWXYZ
FSGNA UDTLB UNKAK HARKG TZVXN	RQPONMLKJIHGFEDCBA
UGBOA JXHFY HTUNH ECTXN OFLSY	J ABCDEFGHIJKLMNOPQRSTUVWXYZ
	QPONMLKJIHGFEDCBA
	K POKMLKJIHGFEDCBAZYXWVUTSRQ
	L ABCDEFGHIJKLMNOPQRSTUVWXYZ
	ONMLKJIHGFEDCBAZYXWVUTSRQ
	M ABCDEFGHIJKLMNOPQRSTUVWXYZ
	NMLKJIHGFEDCBAZYXWVUTSRQ
	N ABCDEFGHIJKLMNOPQRSTUVWXYZ
	MLKJIHGFEDCBAZYXWVUTSRQ
	O ABCDEFGHIJKLMNOPQRSTUVWXYZ
	LKJIHGFEDCBAZYXWVUTSRQ
	P ABCDEFGHIJKLMNOPQRSTUVWXYZ
	KJIHGFEDCBAZYXWVUTSRQ
	Q ABCDEFGHIJKLMNOPQRSTUVWXYZ
	JHGFEDCBAZYXWVUTSRQ
	R ABCDEFGHIJKLMNOPQRSTUVWXYZ
	IHGFEDCBAZYXWVUTSRQ
	S ABCDEFGHIJKLMNOPQRSTUVWXYZ
	HGFEDCBAZYXWVUTSRQ
	T ABCDEFGHIJKLMNOPQRSTUVWXYZ
	GFEDCBAZYXWVUTSRQ
	U ABCDEFGHIJKLMNOPQRSTUVWXYZ
	FEDCBAZYXWVUTSRQ
	V ABCDEFGHIJKLMNOPQRSTUVWXYZ
	EDCBAZYXWVUTSRQ
	W ABCDEFGHIJKLMNOPQRSTUVWXYZ
	DCBAZYXWVUTSRQ
	X ABCDEFGHIJKLMNOPQRSTUVWXYZ
	CBAZYXWVUTSRQ
	Y ABCDEFGHIJKLMNOPQRSTUVWXYZ
	BAZYXWVUTSRQ
	Z ABCDEFGHIJKLMNOPQRSTUVWXYZ
	AZYXWVUTSRQ

# Ingredienti

OTP si basa sulla somma in modulo di un messaggio  $m$  con una chiave  $k$  (entrambe su un dato alfabeto di simboli).

Affinchè OTP funzioni e sia possibile considerarlo perfetto, la chiave  $k$  deve essere:

- completamente **casuale**,
- lunga **almeno quanto** il messaggio,
- **non riutilizzabile**,
- **segreta e condivisa** tra le parti.

# Come funziona OTP?

Ad ogni lettera viene associato un numero.

A	B	C	D	E	F	G	H	I	J	K	L	M	N
0	1	2	3	4	5	6	7	8	9	10	11	12	13

O	P	Q	R	S	T	U	V	W	X	Y	Z
14	15	16	17	18	19	20	21	22	23	24	25

*Crittazione:* Le lettere del messaggio e della chiave, vengono trasformate una a una negli indici corrispondenti  $i_m$  e  $i_k$ . Le lettere vengono poi crittate usando la funzione:

$$i_c \equiv i_m + i_k \quad (26)$$

Si torna infine alla lettera corrispondente all'indice  $i_c$ .

Come possiamo decrittare il messaggio  $m$ ?

# Come funziona OTP?

Ad ogni lettera viene associato un numero.

A	B	C	D	E	F	G	H	I	J	K	L	M	N
0	1	2	3	4	5	6	7	8	9	10	11	12	13

O	P	Q	R	S	T	U	V	W	X	Y	Z
14	15	16	17	18	19	20	21	22	23	24	25

*Decrittazione:* Le lettere del crittogramma e della chiave, vengono trasformate negli indici corrispondenti  $i_c$  e  $i_k$ . Le lettere vengono poi crittate usando la funzione:

$$i_m \equiv i_c - i_k \quad (26)$$

Si torna infine alla lettera corrispondente all'indice  $i_m$ .

# One Time Pad (OTP) - Esempio

A	B	C	D	E	F	G	H	I	J	K	L	M	N
0	1	2	3	4	5	6	7	8	9	10	11	12	13

O	P	Q	R	S	T	U	V	W	X	Y	Z
14	15	16	17	18	19	20	21	22	23	24	25

## *# crittazione*

Testo in chiaro: C I A O | 2 8 0 14

Chiave: A J R F | 0 9 17 5

-----

Testo cifrato: C R R T | 2 17 17 19

## *# decrittazione*

Testo cifrato: C R R T | 2 17 17 19

Chiave: A J R F | 0 9 17 5

-----

Testo in chiaro: C I A O | 2 8 0 4

# Un'Introduzione *Hands-On* a OTP

Come fare:

- si usano due rotoli di carta igienica e due pennarelli,
- incontrarsi in un luogo segreto e scrivere una chiave **segreta** su ciascuno dei due rotoli (una lettera per ogni strappo).

Proviamo! 😊





# OTP - Perché è perfetto?

La *perfezione* di OTP si dimostra con un semplice ragionamento probabilistico.

**Ipotesi:** la chiave di OTP è scelta in maniera completamente casuale.

Supponiamo di avere una versione di OTP che, per crittare un bit  $m_i \in \{0, 1\}$  del messaggio, calcola lo  $\otimes$  (*xor*, somma binaria) tra  $m_i$  e il corrispondente bit della chiave  $k_i$ .

Si ottiene dunque:

$$c_i = m_i \otimes k_i \quad \forall i \in 1, \dots, n$$

Se Eva, la crittoanalista, osserva il passaggio del crittogramma  $c$  sul canale di comunicazione tra Alice e Bob, con quale probabilità può risalire al messaggio originale?

Ovvero, qual è la probabilità che un dato crittogramma  $c$  corrisponda al messaggio  $m$ ?

## OTP - Perché è perfetto? (2)

Usiamo due variabili aleatorie per definire il messaggio  $M$  e il crittogramma  $C$ .

$$P\{M = m|C = c\} = \frac{P\{M = m \wedge C = c\}}{P\{C = c\}}$$

Dato che la chiave  $K$  per cifrare  $m$  è scelta a caso e che è possibile ottenere  $c$  da un certo  $m$  con una e una sola chiave  $k$ . Abbiamo, con  $n$  lunghezza della chiave:

$$P\{K = k\} = \frac{1}{2^n}$$

Dunque, gli eventi  $M = m$  e  $C = c$  sono **indipendenti**. E si conclude che:

$$P\{M = m|C = c\} = \frac{P\{M = m \wedge C = c\}}{P\{C = c\}} = \frac{P\{M = m\}P\{C = c\}}{P\{C = c\}} = P\{M = m\}$$

# Creare Chiavi Casuali (Codice)

Per scrivere il codice di OTP, iniziamo col definire una funzione che ci consente di creare chiavi casuali che ha bisogno di:

```
import random
```

La funzione genera stringhe alfabetiche casuali di lunghezza `length`:

```
def create_otp_key(length, seed=0):  
    key = ''  
    random.seed(seed)  
    for _ in range(length):  
        key += random.choice(list(string.ascii_uppercase))  
    return key
```

# One Time Pad (Codice)

```
def otp_encrypt(msg, key):  
    result = ''  
    msg=msg.upper()  
  
    #TODO: Completare la funzione  
  
    return result  
  
def otp_decrypt(msg, key):  
    result = ''  
    msg=msg.upper()  
  
    #TODO: Completare la funzione  
  
    return result
```

# One Time Pad (Codice)

```
def otp_encrypt(msg, key):  
    result = ''  
    msg=msg.upper()  
    for i in range(len(msg)):  
        c = from_char_to_value(msg[i])  
        k = from_char_to_value(key[i])  
        result += from_value_to_char((c + k) % 26)  
    return result  
  
def otp_decrypt(msg, key):  
    result = ''  
    msg=msg.upper()  
    for i in range(len(msg)):  
        c = from_char_to_value(msg[i])  
        k = from_char_to_value(key[i])  
        result += from_value_to_char((c - k) % 26)  
    return result
```

# One Time Pad (Esempio)

```
msg = "CIAO"  
key = "AJRF"  
  
cifrato = otp_encrypt(msg, key)  
print("Messaggio cifrato: ", cifrato)  
  
inchiaro = otp_decrypt(cifrato, key)  
print("Messaggio decifrato: ", inchiaro)
```

## Esercizio 2

Dove si incontreranno Eva ed Eustachio per iniziare la loro prossima missione segreta?

**CSRM HPYN SPD BEPS GB KFIN, LKB QYAXH N FYFQDUYWSMF, QDW  
YOE RKABXA TGU HRSVFTQDJT MZ IREDZ, MLLCC T EOFP J F ZJTUK.**

data la chiave:

**MYNBIQPMZJPLSGQEJEYDTZIRWZTEJDXCVKPRDLNKTUGRPOQIBZRA  
CXMWZVUATPKHXKWCGSHHZEZROCCKQPDJRJWDRKRGZTRSJOCTZ  
MKSHJFGFBTVIPCC**



# One Time Pad (Byte-wise)

Si tratta di una versione simmetrica del cifrario, dove crittazione e decrittazione si fanno con la stessa funzione:

```
def otp(msg, key):  
    result = ''  
    msg=msg.upper()  
    for i in range(len(msg)):  
        result += chr(ord(msg[i]) ^ ord(key[i]))  
    return result  
  
key = create_key(len(msg))  
  
cifrato = otp(msg, key)  
print("Messaggio cifrato: ", cifrato)  
print("Messaggio decifrato: ", otp(cifrato,key))
```

^ esegue lo XOR tra byte del messaggio e byte della chiave.

ord(c) e chr(v) trasformano un carattere c nel corrispondente codice ASCII e viceversa.



# Diffie-Hellmann (Ripasso)

## programma DH\_ALICE

// Alice sceglie una coppia  $p, g$  nota a tutti  
// e la comunica a Bob

1. **comunica** a Bob:  $p, g$ ;
2. **attende** da Bob: OK;
3. **sceglie** un numero casuale  $a$ , con  $2 \leq a \leq p - 2$ ;
4. **calcola**  $A \leftarrow g^a \bmod p$ ;
5. **comunica** a Bob:  $A$ ;
6. **attende** da Bob:  $B$ ;  
// sarà  $B = g^b \bmod p$ ;  $b$  è il segreto di Bob
7. **calcola**  $K_A \leftarrow B^a \bmod p$ ; // risulta  $K_A = g^{b \cdot a} \bmod p$

## programma DH\_BOB

1. **attende** da Alice:  $p, g$ ;
2. **comunica** ad Alice: OK;
3. **sceglie** un numero casuale  $b$ , con  $2 \leq b \leq p - 2$ ;
4. **calcola**  $B \leftarrow g^b \bmod p$ ;
5. **comunica** ad Alice:  $B$ ;
6. **attende** da Alice:  $A$ ;  
// sarà  $A = g^a \bmod p$ ;  $a$  è il segreto di Alice
7. **calcola**  $K_B \leftarrow A^b \bmod p$ ; // risulta  $K_B = g^{a \cdot b} \bmod p$

# Esponenziazioni Veloci

Una delle parti cruciali dell'algoritmo DH riguarda il tempo di calcolo del valore di  $x^n$  nelle varie fasi dell'algoritmo.

Il metodo si basa sulla regola *ricorsiva*:

$$x^n = x(x^2)^{\lceil \frac{n-1}{2} \rceil} \quad \text{se } n \text{ dispari}$$

e

$$x^n = (x^2)^{\lceil \frac{n}{2} \rceil} \quad \text{se } n \text{ pari}$$

col caso base che  $x^0 = 1$  e  $x^1 = x$ , e considerando che per  $x < 0$  si ha che  $x^n = \frac{1}{x^{-n}}$ .

Questa operazione esegue al più  $\lfloor \lg n \rfloor = O(\lg n)$  passaggi invece di  $O(n)$  necessari per eseguire l'esponenziazione classica per moltiplicazioni successive.

# Esercizio 3

Si scriva il codice di `quadrature_successive(n, x)` e si confronti con l'esponenziazione classica per moltiplicazioni successive, come riportato qui sotto.

```
import time

x, n = 10000, 100000

start = time.time()
quadrature_successive(x, n)
stop = time.time()

print("Quadrature Successive (s): ", stop-start)

start = time.time()
result = 1
for i in range(n):
    result = result * x
stop = time.time()

print("Moltiplicazioni Successive (s): ", stop-start)
```

# Esponenziazioni Veloci (Codice)

```
import math

def quadrature_successive(x, n):
    if n < 0:
        return quadrature_successive(1/x, -n)
    elif n == 0:
        return 1
    elif n == 1:
        return x
    elif n % 2 == 0:
        return quadrature_successive(x*x, math.ceil(n/2))
    elif n % 2 == 1:
        return x * quadrature_successive(x*x, math.ceil((n-1)/2))
```

# Esponenziazioni Veloci (Tempi)

Risultati dei tempi di esecuzione con  $x$ ,  $n = 10000, 100000$  sulla nostra macchina:

```
Quadrature Successive (s): 0.35705089569091797  
Moltiplicazioni Successive (s): 5.113025665283203
```