

Cryptography

Hands-on session

Gaspare Ferraro
ferraro@gaspa.re

November 4, 2020



Visit us!



@GaspareG

Part I

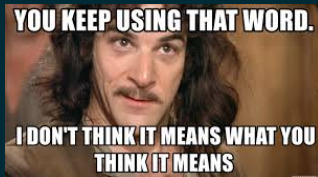
Introduzione

Disclaimer 1

Crypto **is not** Cryptocurrency

CRYPTO IS NOT CRYPTOCURRENCY ๔_๔
IT REFERS TO CRYPTOGRAPHY

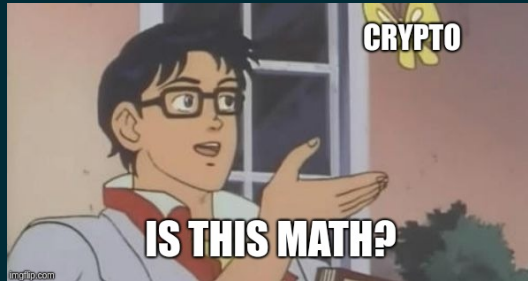
<https://www.cryptoisnotcryptocurrency.com/>



crittografia = kryptós + graphía (let. *scrittura nascosta*)

Disclaimer 2

In questo incontro si fa uso della *matematica*!



Un passo indietro nel tempo...

La crittografia antica



(a) Cifrario di Cesare



(b) Scitala

La crittografia meccanica



Enigma: macchina elettro-meccanica per cifrare e decifrare messaggi usata dai nazisti durante la seconda guerra mondiale.

La crittografia oggi

Le necessità, così come le risorse a disposizione, si sono evolute ed oggi possiamo suddividere la crittografia in:

(EN|DE)CRYPTION

ASYMMETRIC (RSA, ECC, ...)

SYMMETRIC (DES, AES, ...)

KEY EXCHANGE

RSA, DH, ECDH, ...

AUTHENTICATION

RSA, DSA, ECDSA, ...

HASHING

MD5, SHA-1, SHA-256, ...

Part II

Crittografia simmetrica

Crittografia simmetrica

I cifrari simmetrici sono quelli dove i messaggi m vengono cifrati e decifrati usando una stessa chiave k , che deve essere nota solo ed esclusivamente alle due parti.

$\mathcal{C}(m, k) = c$ (funzione di cifratura)

$\mathcal{D}(c, k) = m$ (funzione di decifratura)

Ovviamente deve valere che:

$\mathcal{D}(\mathcal{C}(m, k), k) = m$ (il messaggio originale non viene alterato durante lo scambio).

Per esempio nel cifrario di Cesare:

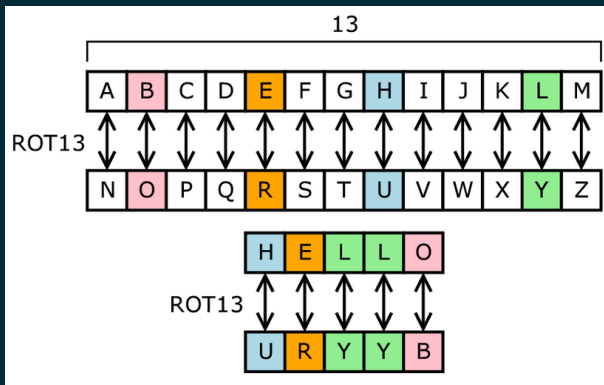
$\mathcal{C}(m, k)$ = ruota in avanti di k ogni singolo carattere.

$\mathcal{D}(c, k)$ = ruota indietro di k ogni singolo carattere.

ROT{13, 47}

ROT13: Cifrario di Cesare con $K = 13$ su alfabeto A-Z.

ROT47: Cifrario di Cesare con $K = 47$ su dizionari ASCII (33 - 126).



Perchè $K = 13$ (or $K = 47$)? Perchè **Encrypt = Decrypt**

Cifrari classici

Cifrari a sostituzione

- Cifrari monoalfabetici: $C_{new} = P[C_{old}]$ (Dove P è una permutazione dell'alfabeto)
(ROT-K è un cifrario monoalfabetico dove P è una rotazione ciclica)
- Cifrari polialfabetici: i caratteri vengono sostituiti usando **molteplici dizionari**

Cifrari a trasposizione

Cifrari dove le **posizioni**, (di caratteri o gruppi di caratteri) del messaggio, sono **trasposti** secondo un sistema ben preciso.

E.g. Vogliamo cifrare il messaggio *WE ARE DISCOVERED. FLEE AT ONCE* usando il **route cipher**:

Griglia:

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

Messaggio: *EJXCTEDECDAEWRIORFEONALEVSE*

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



The screenshot shows the dcode.fr website interface. On the left, a search bar contains the text 'e.g. type scrabble' and a 'GO' button. Below the search bar, the results for 'DCODE' are displayed, showing various tools like 'CTFCAESARCHIPER', 'PGSPNRFNEPUVCRE', etc. On the right, the 'CAESAR CIPHER' tool is highlighted. It includes a 'Sponsored ads' section for 'Caesar Cipher Decoder' and a 'Brute-Force' section with a 'DECRYPT CAESAR CODE' button. The interface is designed to look like old parchment.

Search for a tool

★ SEARCH A TOOL ON DCODE BY KEYWORDS:

e.g. type scrabble GO

Results

Brute-Force : all shifts are tested, text is limited to the first 250 characters.
To keep punctuation and space, please indicate the correct shift found
(+XX).

11	11
+3	CTFCAESARCHIPER
+16	PGSPNRFNEPUVCRE
+18	NEQNLPLDCNSTAPC
+17	OFROMQEMDOTUBQD
+1	EVHECGUCTEJKRGT
+2	DUGDBFTBSDIJQFS
+15	QHTQOSGOFQVWDSF
+14	RIURPTHGRWSETG

CAESAR CIPHER

Cryptography > Substitution Cipher > Caesar Cipher

Sponsored ads

Caesar Cipher Decoder

★ CAESAR SHIFTED CIPHERTEXT

FWI{fdhvdv_kdshu}

KNOWING THE SHIFT: -10

TEST ALL POSSIBLE SHIFTS (BRUTE-FORCE ATTACK)

DECRYPT CAESAR CODE

ROT Cipher — Shift Cipher

With a custom alphabet

★ ALPHABET ABCDEFGHIJKLMNOPQRSTUVWXYZ

★ USE THE ASCII TABLE AS ALPHABET

DECRYPT

Quasi tutti i possibili cifrari classici (e non), vecchi e nuovi, encoder/decoder, ...

XOR cipher

Consideriamo l'operazione XOR \oplus (or esclusivo), valgono le seguenti proprietà:

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

Definiamo lo XOR cipher come:

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

$$\mathcal{D}(c, k) = c \oplus k$$

Problema: la chiave k potrebbe essere più corta del messaggio m .

Soluzione: usiamo ripetutamente la chiave: $k' = k \cdot k \cdot \dots \cdot k$ fino a raggiungere (o superare) la lunghezza di m .

Esempio:

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

$k = 01111000 \ 01111000 \ 01111000 \ 01111000$ (x in ascii 4 volte)

$c = 00011011 \ 00010001 \ 00011001 \ 00010111$ (non stampabile, GxEZFw== in b64)

One-time Pad

Il problema dello XOR cipher è che cifrare usando ripetutamente la stessa chiave può far trapelare delle informazioni *statistiche* sul messaggio originale.

Parliamo quindi di Cifrario di Vernam (o one-time pad) quando la lunghezza della chiave è uguale a quella del messaggio, questo cifrario è chiamato *perfetto* perchè vale:

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

Ovvero la probabilità che M sia un certo messaggio sapendo che il cifrato è C è uguale alla probabilità che M sia un certo messaggio non sapendo il cifrato (tutti i messaggi sono equiprobabili, il messaggio cifrato non ci fornisce informazioni sulla chiave usata).

Bello in teoria, ma:

- ▶ La chiave deve essere scambiata usando un metodo sicuro (scambiarle *a mano*).
- ▶ La chiave deve essere generata casualmente e non riusata (altrimenti è possibile un many-time pad attack).

Crittoanalisi statistica

Spesso la vulnerabilità non è nell'algoritmo ma nella sua applicazione...

- ▶ La chiave è troppo corta rispetto al messaggio
- ▶ La chiave viene ripetuta svariate volte per cifrare diversi messaggi
- ▶ I messaggi usano un dizionario mal distribuito.
- ▶ Conosciamo il formato del messaggio (es: `flag{...}`)

In particolare parliamo di crittoanalisi statistica quando forziamo il cifrario non dal punto di vista algoritmico ma da quello statistico.

Ad esempio in italiano il 33% delle lettere usate è una tra le vocali a, e, i mentre solo con probabilità dello 0.5% si tratterà di una z o una q.

Crittoanalisi statistica

Strumento comodo per l'analisi statistica dei messaggi cifrati:

```
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!
```

Conoscendo la parte iniziale si intravedono delle parole in chiaro:

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

Andando a tentativi si ricostruire la flag finale:

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


Many-Time Pad Interactive

<https://github.com/CameronLonsdale/MTP>

```
Decryptions
1 We can aac or he num er wi tu mputer We can also factor the number w th a
   tra n to ba t re me Ro er Ha
2 Eu er whul pr bably njo tha is orem b ome a corner stone of crypto n nymou
   Eu e theo m
3 Th nicb t ing about eey q i e tograp rs an drive a lot of fancy ca an Bo
4 Th cipoe r ext produc d b a w ry n algo thm looks as good as ciphertex o uced
   sto encry io lg itm Pil p Z r a n
5 Yo don t ant to buy a s of ys m a gu who specializes in stealing ca arc R ber
   menti o li er
6 Th re aue wo ypes o cr tog t which ll eep secrets safe from your t e sis an
   t whi w l k p e et s fe o r e br u i
7 Th re aue wo ypes o cy ogr ne t allo th Government to use brute f o bre he
   o and e h t qu r t e ove n o t
8 We can tee the point her the s ppy if wr ng bit is sent and consume r powe om
   h nviro en A S a r
9 A privfte key encr pti sc at algor hms namely a procedure for ge t ng ke a
   po ure f e c yp ng d p oce o d y z
10 T e Coici e O ford Di tio ry de es cry o a the art of writing o r s n code
11 Th secuet mes age is Wh us tr cipher nev r use the key more than on

Key
6639 89c9dbd8cb9874 2acd63 102eafce78aa ed28a0 c98d29 f8aa 708f80c066c7 f01231
cdd8e802d05ba98777335daefcecd59c433a6b268b60bf4ef03c bb 9a3161edc7 22cfd2 d2 376edba8
c2 027c 24 e2a1 4502 50 alba 2578 911100 e9 02 c4ef a9 8a c083 67
9e 4c
```

"MTP Interactive uses automated cryptanalysis to present a partial decryption which can be solved interactively."

I principi di Shannon

Come valutiamo se un cifrario è abbastanza robusto? (Dove con robustezza è intesa la sua possibilità di essere attaccato con successo).

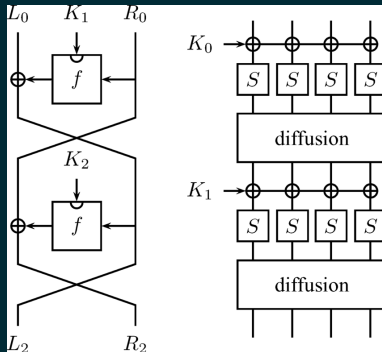
Shannon definisce due concetti chiave:

- ▶ Confusione: la chiave deve essere ben distribuita nel cifrato (ogni bit del cifrato dovrebbe dipendere da ogni bit della chiave).
- ▶ Diffusione: il messaggio deve essere ben distribuito nel cifrato (ogni bit del cifrato dovrebbe dipendere da ogni bit del messaggio).

Nel caso del cifrario di Cesare non abbiamo nessun tipo di diffusione e una bassa confusione (perchè?).

DES & AES

Data Encryption Standard (DES) e Advanced Encryption Standard (AES) si basano sul concetto della *S-Box* (scatola della sostituzione).



La confusione e la diffusione vengono implementate effettuando un (elevato) numero di operazioni invertibili.

DES, 2DES, 3DES

L'algoritmo DES, pubblicato nel 1975, utilizza una chiave a *solo* 56 bit.

Rimase standard fino al 1999, quando un gruppo di ricercatori riuscì a rompere una chiave di crittazione in sole 22 ore.

Con la potenza di calcolo odierna basterebbero poche ore.

Prima di introdurre l'AES un'alternativa più *robusta* (ed immediata) all'ormai violato algoritmo DES fu quella di ripetere il DES 3 volte (algoritmo 3DES).

Perchè 3DES e non 2DES?

Usando 2 chiavi a 56 bit mi aspetterei la stessa sicurezza di una chiave a 112 bit, ma non è così...



Attacco Meet-In-The-Middle

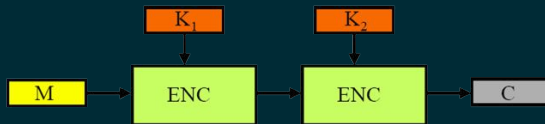
$$C = 2DES_{enc}(M, K_1, K_2) = DES_{enc}(DES_{enc}(M, K_2), K_1)$$

$$C_1 = DES_{enc}(M, K_1)$$

$$C_2 = DES_{enc}(C_1, K_2)$$

Come possiamo trovare le chiavi conoscendo M e C ?

Con un attacco **Meet-In-The-Middle**



Provo tutte le chiavi k_1 per generare $C_1 = DES_{enc}(M, K_1)$

Provo tutte le chiavi k_2 per generare $C_1 = DES_{dec}(C_2, K_2)$

Quando trovo una collisione tra il primo e il secondo C_1 ho trovato la coppia (k_1, k_2) .

Risultato: 2 chiavi a 56 bit proteggono quanto una chiave a 57 bit.

Attack models

Classificazione degli attacchi in crittografia:

- ▶ **Known-plaintext** attack: Accesso ad un numero limitato di coppie messaggio/cifrato
- ▶ **Ciphertext-only** attack: Accesso solo al messaggio cifrato.
- ▶ **Chosen plaintext** attack: Possibilità di scegliere messaggi da cifrare (encrypt oracle)
- ▶ **Chosen ciphertext** attack: Possibilit di scegliere messaggi cifrati da decifrare (decrypt oracle)
- ▶ **Side-channel** attack: Uso di informazioni "esterne" per rompere il cifrario (tempo, suono, errori, ...)

Part III

Crittografia asimmetrica

Concetti base

La crittografia asimmetrica si basa sulle funzioni *one-way trapdoor* e sulla presenza di una coppia di chiavi (chiamate chiave pubblica e chiave privata).

Una funzione f si dice *trapdoor* se:

- ▶ Calcolare $y = f(x)$ è computazionalmente facile.
- ▶ Calcolare $x = f^{-1}(y)$ è computazionalmente difficile (*senza nessuna informazione aggiuntiva*).

Un esempio è il problema della fattorizzazione:

- ▶ $m = f(\{p, q\}) = (p * q)$ (calcolo del prodotto tra i primi p e q).
- ▶ $\{p, q\} = f^{-1}(m) = ??$ (scomposizione in fattori primi di n).

$f(\{49171, 61843\}) = 3040882153$ (facile quanto aprire una calcolatrice).

$f^{-1}(1841488427) = ??$ (devo provare tutti i divisori da 2 a \sqrt{n}).

Il problema diventa banale se conosco uno dei due divisori:

$$1841488427 / 58049 = 31723$$

Aritmetica modulare

Diciamo che due interi a e b sono congrui modulo n , scritto $a \equiv b \pmod{n}$, se $(a \% n) = (b \% n)$ dove $\%$ è il resto della divisione intera (modulo).

Alcune proprietà matematiche:

- ▶ $a + k \equiv b + k \pmod{n}$, invariante per addizione.
- ▶ $k * a \equiv k * b \pmod{n}$, invariante per moltiplicazione.
- ▶ $a^k \equiv b^k \pmod{n}$, invariante per potenza.
- ▶ $\sqrt{a} \equiv b \pmod{n}$ se $a \equiv b^2 \pmod{n}$, radice quadrata.
- ▶ $a^{-1} \equiv b \pmod{n}$ se $ab \equiv 1 \pmod{n}$, inverso moltiplicativo.
- ▶ ...

RSA pt.1

L'algoritmo di cifratura asimmetrica più famoso è l'RSA (da Rivest Shamir Adleman) che si basa sul problema della fattorizzazione e sull'aritmetica modulare.

Il funzionamento di base è:

- ▶ Si scelgono due numeri primi a caso p e q (in modo *sicuro*).
- ▶ Si calcola il prodotto $n = p * q$ (che sarà il nostro modulo).
- ▶ Si calcola il toziente $\phi(n) = (p - 1) * (q - 1)$ (il numero dei coprimi con n).
- ▶ Si sceglie a caso un numero e , coprimo e minore di $\phi(n)$.
- ▶ Si calcola il numero d come inverso moltiplicativo di e , ovvero $ed \equiv 1 \pmod{\phi(n)}$.

Chiamiamo quindi:

$k_{pub} = (n, e)$ la chiave pubblica che distribuiamo.

$k_{priv} = (n, d)$ la chiave privata che teniamo segreta.

RSA pt.2

Una volta calcolata la chiave pubblica e quella privata possiamo cifrare e decifrare i messaggi in questo modo:

$$\mathcal{C}(m, k_{pub}) = m^e \pmod{n}$$

$$\mathcal{D}(c, k_{priv}) = c^d \pmod{n}$$

Si ma perchè funziona?

Dal teorema di Eulero sappiamo che $m^{\phi(n)} \equiv 1 \pmod{n}$.

I valori e e d sono calcolati in modo che $ed \equiv 1 \pmod{\phi(n)}$.

$$\mathcal{D}(\mathcal{C}(m, k_{pub}), k_{priv}) = m^{ed} = m^{\phi(n)*t+1} = m^{\phi(n)*t} * m = 1 * m = m$$

Un esempio per capire

Scegliamo i seguenti parametri:

- ▶ $p = 13$ e $q = 23$
- ▶ $n = p * q = 299$
- ▶ $\phi(n) = (13 - 1) * (23 - 1) = 264$
- ▶ $e = 7$, $\gcd(264, 7) = 1$
- ▶ $d = 151$, $7 * 151 \equiv 1 \pmod{264}$

Quindi $k_{pub} = (299, 7)$ e $k_{priv} = (299, 151)$. Vogliamo cifrare $m = 42$.

$$\mathcal{C}(m, k_{pub}) = m^e = 42^7 = 230539333248 = 107 \pmod{299}$$

$$\mathcal{D}(c, k_{priv}) = c^d = 107^{151} = 2743956545...948643 = 42 \pmod{299}$$

(Non tanto) a caso

"Si scelgono due numeri primi *a caso* p e q (in modo *sicuro*)"

- ▶ Scegliere p e q di almeno 1024 bit.
- ▶ Scegliere p e q non troppo vicini tra loro.
- ▶ Non riutare uno dei primi per altri moduli.

Con la potenza di calcolo attuale è possibile fattorizzare semiprimi fino a (circa) 768 bit, i moduli più grandi sono (per ora) resistenti agli attacchi bruteforce.

Se p e q sono vicini allora abbiamo che $n \simeq p^2 \simeq q^2$ e quindi anche \sqrt{n} sarà vicino ai primi. Basterà quindi un attacco bruteforce che cerca i fattori vicino alla radice quadrata.

Se $n_1 = p * q'$ e $n_2 = p * q''$ allora $p = \gcd(n_1, n_2)$.

OpenSSL per l'RSA

OpenSSL è un'implementazione open source dei protocolli SSL e TLS. OpenSSL implementa e fornisce i tools per i più importanti protocolli crittografici.

Generare un numero primo a n bits:

```
OPENSSL PRIME -GENERATE -BITS 64
```

Generare una chiave RSA a n bits:

```
OPENSSL GENRSA -OUT EXAMPLE.KEY 512
```

Stampare i parametri della chiave privata:

```
OPENSSL RSA -IN EXAMPLE.KEY -TEXT -NOOUT
```

Stampare chiave pubblica da chiave privata:

```
OPENSSL RSA -IN EXAMPLE.KEY -PUBOUT
```

Stampare il modulo da chiave privata:

```
OPENSSL RSA -IN EXAMPLE.KEY -NOOUT -MODULUS
```

Attacchi all'RSA

<https://github.com/Ganapati/RsaCtfTool>

Implementazione dei più classici attacchi all'RSA, tipo:

- Weak public key factorization
- Small q (q less than 100,000)
- Fermat's factorisation for close p and q
- Past CTF Primes method
- Common factor attacks across multiple keys

Trovare la chiave privata:

```
./RsaCtfTool.py -PUBLICKEY ./KEY.PUB -PRIVATE
```

Dumpare i parametri da una chiave:

```
./RsaCtfTool.py -DUMPKEY -KEY ./KEY.PUB
```

Decifrare file:

```
./RsaCtfTool.py -PUBLICKEY ./KEY.PUB -UNCIPHERFILE ./CIPHERED_FILE
```

Romperere multiple chiavi pubbliche con fattori comuni:

```
./RsaCtfTool.py -PUBLICKEY "*.PUB" -PRIVATE
```

Part IV

Hashing

Creare confusione

Chiamiamo hash una funzione f *non invertibile* che associa una stringa ad una stringa di dimensione fissata. Con le seguenti proprietà:

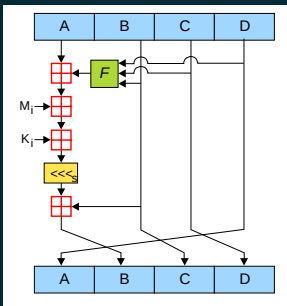
- ▶ Resistenza alla preimmagine: dato un hash h è difficile trovare m tale che $f(m) = h$.
- ▶ Resistenza alla seconda preimmagine: dato m_1 è difficile trovare m_2 tale che $f(m_1) = f(m_2)$.
- ▶ Resistenza alle collisioni: è difficile trovare coppie (m_1, m_2) tali che $f(m_1) = f(m_2)$.

Utilizzi pratici:

- ▶ Salvataggio delle password (hashate invece che in chiaro).
- ▶ Creazione di hashtable (struttura dati ad accesso diretto).
- ▶ Controllo di integrità (*md5sum*, *sha1sum*).
- ▶ Firma digitale (sull'hash invece che sul documento).

MD5

Inventato da Rivest nel 1991, divenne la funzione di hash standard fino al 2004-2006. Viene utilizzata ancora adesso per il controllo di integrità.



$\text{md5}(\text{"password"}) = 5\text{F}4\text{DCC}3\text{B}5\text{AA}765\text{D}61\text{D}8327\text{DEB}882\text{CF}99$

$\text{md5}(\text{"passwore"}) = \text{A}826176\text{C}6495\text{C}5116189\text{DB}91770\text{E}20\text{CE}$

Proof of Work

Spesso, come sistema di protezione per attacchi DOS o per altri motivi, ci viene chiesto di reversare (completamente o parzialmente) un hash.

L'accesso ad una risorsa ci verrà fornito solo dopo aver dimostrato (*Proof of Work*) la risoluzione ad un problema. Nel caso dell'hashing questa operazione viene chiamata hashcash (ed è alla base del mining delle criptovalute).

Esempio (VolgaCtf2017):

Solve a puzzle: find an x such that 26 last bits of $\text{SHA1}(x)$ are set, $\text{len}(x)==29$ and $x[:24]==\text{'58a5a7950d2ec81fae5c1c74'}$

Trovare le collisioni

Come trovo una collisione / preimmagine?

- ▶ Database online (es. crackstation.net), immediato ma non completo.
- ▶ Bruteforce (es. John The Ripper, HashCat), lento ma completo.

Quanti tentativi dovrò fare per reversare un hash?

Paradosso dei compleanni:

se la funzione hash restituisce un output da n bit ($m = 2^n$ possibili hash) si avrà una collisione al 50% di probabilità dopo $2^{(n/2)}$ tentativi (\sqrt{m}).

Crackare gli zip: *fcrackzip*

FCRACKZIP è un tool per il *recupero* di password smarrite di archivi compressi.

```
pentaroot@kali:~$ fcrackzip --help

fcrackzip version 1.0, a fast/free zip password cracker
written by Marc Lehmann <pcg@goof.com> You can find more info on
http://www.goof.com/pcg/marc/

USAGE: fcrackzip
      [-b|--brute-force]      use brute force algorithm
      [-D|--dictionary]      use a dictionary
      [-B|--benchmark]       execute a small benchmark
      [-c|--charset charset] use characters from charset
      [-h|--help]            show this message
      [--version]            show the version of this program
      [-V|--validate]        sanity-check the algorithm
      [-v|--verbose]         be more verbose
      [-p|--init-password string] use string as initial password/file
      [-l|--length min-max]  check password with length min to max
      [-u|--use-unzip]        use unzip to weed out wrong passwords
      [-m|--method num]      use method number "num" (see below)
      [-2|--modulo r/m]      only calculate 1/m of the password
      file...                the zipfiles to crack

methods compiled in (* = default):

  0: cpmask
  1: zip1
  *2: zip2, USE_MULT_TAB

pentaroot@kali:~$
```

Comando tipico: *fcrackzip -b -u -v -l 3-4 prova.zip*

Trovare le collisioni

John The Ripper

```
root@kali: ~  
File Edit View Search Terminal Help  
root@kali:~# john -format=LM /root/Desktop/hash.txt  
Using default input encoding: UTF-8  
Using default target encoding: CP850  
Loaded 4 password hashes with no different salts (LM [DES 128/128 AVX-16])  
Remaining 3 password hashes with no different salts  
Press 'q' or Ctrl-C to abort, almost any other key for status  
0g 0:00:01:21 0.03% 3/3 (ETA: 2017-05-16 00:22) 0g/s 29892Kp/s 29892Kc/s 90718KC  
/s 085MSYP..085MSNY  
0g 0:00:01:22 0.03% 3/3 (ETA: 2017-05-16 00:14) 0g/s 29957Kp/s 29957Kc/s 90903KC  
/s FCUCDB7..FCUCDGT  
0g 0:00:01:25 0.03% 3/3 (ETA: 2017-05-16 00:00) 0g/s 30073Kp/s 30073Kc/s 91218KC  
/s NEWSAHC..NEWSB9B  
0g 0:00:01:26 0.03% 3/3 (ETA: 2017-05-15 23:54) 0g/s 30114Kp/s 30114Kc/s 91330KC  
/s VELH10..VELHLS  
0g 0:00:01:27 0.04% 3/3 (ETA: 2017-05-15 23:50) 0g/s 30148Kp/s 30148Kc/s 91423KC  
/s 4A93GP..4A902K  
0g 0:00:01:28 0.04% 3/3 (ETA: 2017-05-15 23:44) 0g/s 30201Kp/s 30201Kc/s 91572KC  
/s 08TS4DA..08TSFOA  
0g 0:00:01:29 0.04% 3/3 (ETA: 2017-05-15 23:37) 0g/s 30254Kp/s 30254Kc/s 91720KC  
/s IKPABAO..IKPABR6  
0g 0:00:01:30 0.04% 3/3 (ETA: 2017-05-15 23:33) 0g/s 30283Kp/s 30283Kc/s 91799KC  
/s 0J0DGB..0J0D5C  
0g 0:00:01:33 0.04% 3/3 (ETA: 2017-05-15 23:19) 0g/s 30401Kp/s 30401Kc/s 92124KC  
/s H10GW8W..H106CL1
```

Trovare le collisioni

Hashcat

```
Session.....: hashcat
Status.....: Running
Hash.Type.....: iTunes backup >= 10.0
Hash.Target.....: $itunes_backup$*10*50782427bcc454da54b51256351cb1de...c32eb7
Time.Started.....: Sun Jan 14 14:15:38 2018 (1 hour, 5 mins)
Time.Estimated...: Sun Jan 14 18:05:56 2018 (2 hours, 44 mins)
Guess.Mask.....: Oliver?a?a [8]
Guess.Queue.....: 2/8 (25.00%)
Speed.Dev.#2.....: 1 H/s (1.84ms)
Recovered.....: 0/1 (0.00%) Digests, 0/1 (0.00%) Salts
Progress.....: 2560/9025 (28.37%)
Rejected.....: 0/2560 (0.00%)
Restore.Point....: 2560/9025 (28.37%)
Candidates.#2....: Oliver>e -> OliverT9
HwMon.Dev.#2.....: N/A

[s]tatus [p]ause [r]esume [b]ypass [c]heckpoint [q]uit ->

Session.....: hashcat
Status.....: Running
Hash.Type.....: iTunes backup >= 10.0
Hash.Target.....: $itunes_backup$*10*50782427bcc454da54b51256351cb1de...c32eb7
Time.Started.....: Sun Jan 14 14:15:38 2018 (1 hour, 13 mins)
Time.Estimated...: Sun Jan 14 18:03:02 2018 (2 hours, 34 mins)
Guess.Mask.....: Oliver?a?a [8]
Guess.Queue.....: 2/8 (25.00%)
Speed.Dev.#2.....: 1 H/s (1.84ms)
Recovered.....: 0/1 (0.00%) Digests, 0/1 (0.00%) Salts
Progress.....: 2816/9025 (31.20%)
Rejected.....: 0/2816 (0.00%)
Restore.Point....: 2816/9025 (31.20%)
Candidates.#2....: OliverPh -> Oliverj6
HwMon.Dev.#2.....: N/A

[sl]tatus [ol]ause [rl]esume [bl]ypass [cl]heckpoint [ql]uit ->
```

Come difendersi (lato client)

Scegliere la giusta password

<p>UNCOMMON (NON-GIBBERISH) BASE WORD</p> <p>ORDER UNKNOWN</p> <p>Tr0ub4dor & 3</p> <p>CAPS? COMMON SUBSTITUTIONS NUMERAL PUNCTUATION</p> <p>(YOU CAN ADD A FEW MORE BITS TO ACCOUNT FOR THE FACT THAT THIS IS ONLY ONE OF A FEW COMMON FORMATS.)</p>	<p>~28 BITS OF ENTROPY</p> <p>$2^{28} = 3 \text{ DAYS AT } 1000 \text{ GUESSES/SEC}$</p> <p>(PLAUSIBLE ATTACK ON A WEAK REMOTE WEB SERVICE: YES. CRACKING A STOKEN MATHS IS FASTER, BUT IT'S NOT WHAT THE INTRUDER DOES. SHOULD WORRY ABOUT.)</p> <p>DIFFICULTY TO GUESS: EASY</p>	<p>WAS IT TROMBONE? NO, TROUBADOR. AND ONE OF THE 0s WAS A ZERO?</p> <p>AND THERE WAS SOME SYMBOL...</p> <p>DIFFICULTY TO REMEMBER: HARD</p>
<p>correct horse battery staple</p> <p>FOUR RANDOM COMMON WORDS</p>	<p>~44 BITS OF ENTROPY</p> <p>$2^{44} = 530 \text{ YEARS AT } 1000 \text{ GUESSES/SEC}$</p> <p>DIFFICULTY TO GUESS: HARD</p>	<p>THAT'S A BATTERY STAPLE.</p> <p>CORRECT!</p> <p>DIFFICULTY TO REMEMBER: YOU'VE ALREADY MEMORIZED IT</p>

THROUGH 20 YEARS OF EFFORT, WE'VE SUCCESSFULLY TRAINED EVERYONE TO USE PASSWORDS THAT ARE HARD FOR HUMANS TO REMEMBER, BUT EASY FOR COMPUTERS TO GUESS.

Usare un token per la 2FA

Come difendersi (lato server)

Basta un poco di sale...

Abbiamo parlato di come *non salvare le password in chiaro* bensì salvare il loro hash.



Problema:

Se più persone usano la stessa password (tipico: *qwerty*, *123456*, ...) avrò hash duplicati nel database.

Soluzione:

Salvare $f(\text{password} + \text{salt})$ invece che $f(\text{password})$, dove *salt* è una stringa casuale generata per persona (possibilmente memorizzata in un luogo separato ai dati di login).

Un (pessimo) esempio...




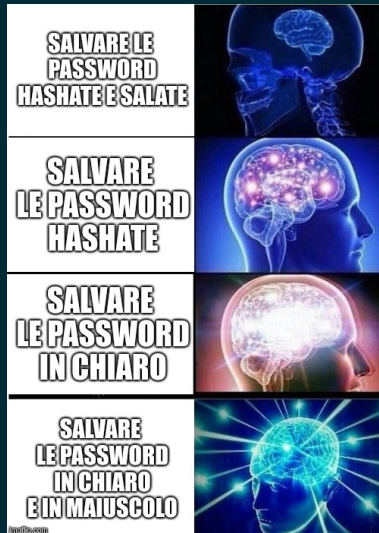
Gentile GASPARE FERRARO,

come da tua richiesta, ecco i dati relativi al tuo Account

Riepilogo dati registrazione

USER-ID	Gasparg
PASSWORD	XZA13254A

www.trenitalia.com



Fine

