Crittografia: password e chiave di criptazione

È un brutto periodo per le password e la security. Sempre più spesso si sente parlare di leaks, furti di password, intrusioni, ecc..
Questo preoccupa molto e si torna sempre a dire di cambiare spesso le password e renderle sempre più complicate. Poi si sente parlare di algoritmi di cifratura a X bit, che sono ultrasicuri, di hash, di salt; ma cosa significa veramente tutta questa forza dell'algoritmo di cifratura?
Partiamo dall'inizio, ossia criptare un messaggio o file o altro. Significa renderlo illeggibile per chiunque non abbia la password per poterlo rendere in chiaro.
I sistemi di cifratura sono di due tipi:

1) a chiave simmetrica, ossia mittente e destinatario conoscono la password
2) A chiave asimmetrica, ossia il mittente cifra con la chiave pubblica del destinatario ed il destinatario decifra con la propria chiave privata, che conosce solo lui.

Ma cerchiamo di capire bene che rapporto c'è tra la chiave di cifratura e la password. proprio perchè si sente parlare di chiavi a 128,

Nanni Bassetti è Laureato in Scienze dell'Informazione a Bari ed è libero professionista specializzato in informatica forense. Ha collaborato come free-lance con molte riviste informatiche nazionali e internazionali e come docente per molti corsi presso enti, scuole e università, ha inoltre scritto articoli divulgativi di programmazione, web usability, sicurezza informatica e digital forensics. Ha lavorato per alcune Procure della Repubblica oltre che come CTU/CTP per molte analisi forensi informatiche civili e penali. Iscritto all'albo dei C.T.U. presso il Tribunale di Bari, è consulente di parte civile per alcuni casi di risonanza nazionale. Fondatore di **CFI – Computer Forensics Italy** – la più grande community di computer forensics italiana e segretario di ONIF (Osservatorio Nazionale Informatica Forense). Project manager di **Caine Linux** Live Distro forense. Curatore del sito **Scripts4cf** dedicato a software per la computer forensics.

256, 1024 bit e così via, che non implicano numeri sempre più elevati di tentativi per poterle trovare, ma se si usa una password debole come "pippo", in pochi secondi chiunque riesce ad aprire il nostro file cifrato con un algoritmo a 256 bit.

CHE SIGNIFICA?

Ho pensato ad un percorso ad esempi per spiegare semplicemente il meccanismo.
Dunqu, iniziamo da una cifratura di un messaggio semplice con un cifrario del 1586, ossia il famoso cifrario di Vigenère (https://it.wikipedia.org/wiki/Cifrario_di_Vigen%C3%A8re), che utilizza una chiave alfabetica per creare il testo criptato semplicemente spostando ogni lettera del testo in chiaro del numero di posti relativo alla lettera corrispondente della chiave di cifratura, che si ripete fino alla fine del testo in chiaro.

NANNI BASSETTI

Esempio:

Testo in chiaro: ciaoailettori **Chiave:** pippo

Si deve posizionare la chiave sotto il testo, ripetendola fine alla fine dello stesso:

C	I	A	O	A	I	L	E	T	T	O	R	I
P	I	P	P	O	P	I	P	P	O	P	I	P

Per cifrare il messaggio basta aggiungere al numero di posizione della lettera del testo in chiaro, il numero di posizione della lettera della chiave e, se supera il 26 (il numero di lettere dell'alfabeto), basta sottrarre 26 e ricavare il numero della lettera che finirà nel testo cifrato. Infine, si consideri che il conteggio delle posizioni inizia da 0 (zero) ossia la lettera A occupa la posizione 0, la B la posizione 1 e così via.

Esempio:

```
ALFABETO:
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
0 1 2 3 4 5 6  7 8 9 10 11  12 13  14 15  16  17  18 19 20  21 22  23 24  25
```

TESTO IN CHIARO	A	B	C
CHIAVE	M	I	K
TESTO CIFRATO	M	J	M

A=0 B=1 C=2
+
M=12 I=8 K=10
12 9 12
Ossia: MJM

Per decriptare:

M=12 J=9 M=12
-
M=12 I=8 K=10
0 1 2
Ossia: ABC

Se da un'addizione (cifratura) si ottiene un numero maggiore di 26, basta effettuare la sottrazione di 26 al quel numero per avere la lettera corrispondente (es. 32 - 26 = 6 ossia la lettera G).

Se da una sottrazione (decifratura) risultasse un numero negativo, basterà aggiungere 26 per riportarlo nel range dei valori dell'alfabeto, es. V=21 - Z=25 = -4, quindi -4 + 26 = 22 che corrisponde alla lettera W.

Se la posizione della lettera di cifratura o di decifratura è 26, si "riavvolge" il nastro. Il 26-esimo elemento non esiste quindi diventa 0 ossia si riparte dalla lettera A.

Ecco qui un piccolo programma in Python che ho realizzato per vedere il cifrario in azione:

```
#!/usr/bin/python2
# VIGENERE_NB.py
# Criptare e decriptare col cifrario di Vigenere
#
# by Nanni Bassetti - http://www.nannibassetti.com - digitfor@gmail.com

alfabeto = 'abcdefghijklmnopqrstuvwxyz'
criptato=''
decriptato=''

def main():
        domanda=raw_input('Vuoi criptare o decriptare? (c/d):').lower()
        if domanda == 'c':
                criptare()
        if domanda == 'd':
                decriptare()
def criptare():
        global criptato
        testo = raw_input('Inserisci un testo da criptare:').lower()
        print 'testo:',testo

        chiave = raw_input('Inserisci un chiave di criptazione:').lower()
        print 'chiave:',chiave
        while len(chiave)<len(testo):
    #controllo se la chiave e' piu' corta del testi
                chiave+=chiave
    #This repeats the chiave.
```

```
                    if len(chiave)>len(testo):
        #controllo se la chiave e' piu' lunga del testo
                            newkey=chiave[:len(testo)]
                            print 'newkey=',newkey
        #qui si taglia la chiave alla lunghezza del testo
            for t,c in zip(testo,chiave):
                    chart = alfabeto.index(t)
                    charc = alfabeto.index(c)
                    newchar = chart + charc
                    if newchar >= 26:
                            newchar -= 26
                    newchar = alfabeto[newchar]
                    criptato+=newchar
            print 'testo criptato: ',criptato

def decriptare():
            global decriptato
            testo = raw_input('Inserisci un testo da decriptare:').lower()
            print 'testo:',testo

            chiave = raw_input('Inserisci un chiave di decriptazione:').lower()
            print 'chiave:',chiave

            while len(chiave)<len(testo):
    #controllo se la chiave e' piu' corta del testi
                            chiave+=chiave
    #This repeats the chiave.
    if len(chiave)>len(testo):
        #controllo se la chiave e' piu' lunga del testo
        newkey=chiave[:len(testo)]
        #qui si taglia la chiave quando finisce il testo
            for t,c in zip(testo,chiave):
                    chart = alfabeto.index(t)
                    charc = alfabeto.index(c)
                    newchar = chart - charc
                    if newchar >= 26:
                            newchar -= 26
                    newchar = alfabeto[newchar]
                    decriptato+=newchar
            print 'testo decriptato: ',decriptato
main()
```

```
C:\[                    \>python vigenere_nb.py
Vuoi criptare o decriptare? (c/d):c
Inserisci un testo da criptare:vivalapuglia
testo: vivalapuglia
Inserisci un chiave di criptazione:pluto
chiave: pluto
newkey= plutoplutopl
testo criptato:  ktptzpaozzxl

C:\                     \>python vigenere_nb.py
Vuoi criptare o decriptare? (c/d):d
Inserisci un testo da decriptare:ktptzpaozzxl
testo: ktptzpaozzxl
Inserisci un chiave di decriptazione:pluto
chiave: pluto
testo decriptato:  vivalapuglia
```

Ok, abbiamo descritto il cifrario di Vignère ed abbiamo capito cos'è la chiave di cifratura, ma in quest'esempio sembra ancora confondersi con la password, pertanto dobbiamo introdurre il concetto di funzione di derivazione della chiave (Key derivation function o KDF) - https://it.wikipedia.org/wiki/Derivazione_di_una_chiave_crittografica
Senza addentrarci nelle complessità matematiche, il concetto è che da un elemento noto, come una password, si può derivare la chiave di cifratura, che potrà anche essere molto ma molto più complessa della password che l'ha generata e quindi un attacco mirato a scoprirla sarebbe veramente difficile e lungo nel tempo.
Per questo si preferisce attaccare la password, che è più facile da trovare sia con gli attacchi di forza bruta sia con gli attacchi a dizionario. Per esempio la password "pippo" in un dizionario è presente ed un computer ci metterà un centesimo di secondo a trovarla e a provare ad aprire il file cifrato. Una volta immessa la password essa genererà la chiave di cifratura che servirà a decriptare il contenuto.

SCHEMA:

PASSWORD –-> KDF (funzione di derivazione della chiave) –> CHIAVE –-> TESTO CIFRATO/DECIFRATO

Quindi, se possiamo generare 100.000 password al minuto possiamo generare 100.000 chiavi al minuto, ma questo sarebbe insensato. Il senso di derivare la chiave è anche nel rallentare l'attacco, ossia si fa in modo che la funzione di derivazione sia complessa; in tal modo per ogni password provata/immessa, possiamo pensare che la funzione impieghi 1 secondo a creare la chiave. Questa restrizione abbasserebbe l'attacco all'immissione di 60 tentativi al minuto e non più 100.000, rendendo il tempo un fattore critico ed economico per l'attaccante. La tecnica per fare questo si chiama **Key Stretching** https://en.wikipedia.org/wiki/Key_stretching, ossia effettuare una serie di elaborazioni tali da rallentare la generazione della chiave anche partendo da password deboli. Per esempio, se una volta che immettiamo la password la chiave di cifratura viene generata calcolando 500.000 volte l'hash della stessa + un salt:

Hash("pluto")=ABD345
Hash("pluto" + 15BA) = FFC881

vuol dire che il programma, prima di comunicarci se abbiamo beccato la password o meno, impiegherà un tempo X per elaborare la chiave, così da rallentare il prossimo tentativo. E' possibile provare cambiando il numero dei cicli, come si allungano i tempi, utilizzando questo prorgamma in Python destritto a tal fine:

```
#!/usr/bin/python2
# sha256_500k.py
# calcolo del tempo per 500000 sha256
#
# by Nanni Bassetti - http://www.nannibassetti.com - digitfor@gmail.com

import hashlib
i=0
from datetime import datetime

tstart = datetime.now()
print "tempo iniziale:", tstart
while i<500000: # qui potete cambiare i cicli per vedere come aumentano/diminuiscono i tempi
        i=i+1
        hash_pw_salt = hashlib.sha256('pippo'+unicode(i))
        hex_value = hash_pw_salt.hexdigest()
        #print(hex_value)
tend = datetime.now()
print "tempo finale : ", tend
print "tempo totale: ",tend - tstart
print 'fine'
print(hex_value)
```

```
C:\...>python sha256_500k.py
tempo iniziale: 2016-09-10 19:02:55.389000
tempo finale :  2016-09-10 19:02:56.985000
tempo totale:  0:00:01.596000
fine
b618b90015994f0bf0657648a469476aa96672eeaa17a9d609b0b98722406351
```

Ora possiamo scrivere un simulatore didattico che utilizzerà una password da voi scelta ma che cifrerà con una chiave da lui generata. Dunque torniamo a Vigenère, ma modificato ad hoc, ossia estendiamo l'alfabeto a 36 caratteri aggiungendo le cifre da 0 a 9 e utilizzando l'hash SHA256 (64 caratteri di lunghezza) come chiave, anche se utilizzerete la password "pippo" in realtà la chiave che Vigenère utilizzerà sarà:

b618b90015994f0bf0657648a469476aa96672eeaa17a9d609b0b98722406351

```
#!/usr/bin/python2
# VIGENERE_stretched.py
# Criptare e decriptare col cifrario di Vigenere
#
# by Nanni Bassetti - http://www.nannibassetti.com - digitfor@gmail.com

import hashlib
encoded=""
alfabeto = 'abcdefghijklmnopqrstuvwxyz0123456789'
criptato=''
decriptato=''
i=0

def main():
        domanda=raw_input('Vuoi criptare o decriptare? (c/d):').lower()
        if domanda == 'c':
                criptare()
        if domanda == 'd':
                decriptare()

def criptare():
        global criptato,i
        testo = raw_input('Inserisci un testo da criptare:').lower()
```

```
        print 'testo:',testo

        pw = raw_input('Inserisci una password di criptazione:').lower()
        print 'Password:',pw
        while i<500000:
                i=i+1
                hash_pw_salt = hashlib.sha256(pw+unicode(i))
                chiave = hash_pw_salt.hexdigest()
        while len(chiave)<len(testo):
    #controllo se la chiave e' piu' corta del testi
                chiave+=chiave
    #This repeats the chiave.
                if len(chiave)>len(testo):
      #controllo se la chiave e' piu' lunga del testo
                        newkey=chiave[:len(testo)]
                        print 'newkey=',newkey
      #qui si taglia la chiave alla lunghezza del testo
        for t,c in zip(testo,chiave):
                chart = alfabeto.index(t)
                charc = alfabeto.index(c)
                newchar = chart + charc
                if newchar >= 36:
                        newchar -= 36
                newchar = alfabeto[newchar]
                criptato+=newchar
        print 'testo criptato: ',criptato

def decriptare():
        global decriptato,i
        testo = raw_input('Inserisci un testo da decriptare:').lower()
        print 'testo:',testo

        pw = raw_input('Inserisci una password di decriptazione:').lower()
        print 'password:',pw
        while i<500000:
                i=i+1
                hash_pw_salt = hashlib.sha256(pw+unicode(i))
                chiave = hash_pw_salt.hexdigest()
        while len(chiave)<len(testo):
    #controllo se la chiave e' piu' corta del testi
                        chiave+=chiave
    #This repeats the chiave.
    if len(chiave)>len(testo):
      #controllo se la chiave e' piu' lunga del testo
      newkey=chiave[:len(testo)]
      #qui si taglia la chiave quando finisce il testo
        for t,c in zip(testo,chiave):
                chart = alfabeto.index(t)
                charc = alfabeto.index(c)
                newchar = chart - charc
                if newchar >= 36:
                        newchar -= 36
                newchar = alfabeto[newchar]
                decriptato+=newchar
        print 'testo decriptato: ',decriptato

main()
```

```
C:\                    >python vigenere_stretched.py
Vuoi criptare o decriptare? (c/d):c
Inserisci un testo da criptare:tantisalutidanannibassetti
testo: tantisalutidanannibassetti
Inserisci una password di criptazione:pippo
Password: pippo
testo criptato:  u6erjr0blohc4s0os875po8rtc
```

```
C:\                    >python vigenere_stretched.py
Vuoi criptare o decriptare? (c/d):d
Inserisci un testo da decriptare:u6erjr0blohc4s0os875po8rtc
testo: u6erjr0blohc4s0os875po8rtc
Inserisci una password di decriptazione:pippo
password: pippo
testo decriptato:  tantisalutidanannibassetti
```

Possiamo quindi concludere che scegliere una password resistente è importante per infastidire gli attacchi brute force e a dizionario, ma è altrettanto importante che la funzione di derivazione sia lenta. Non fidiamoci solo dei numeri altisonanti, "algoritmo a 128 bit!" perché, se non è implementato bene, la cifratura sarà fatta con una chiave a 128 bit, ma se il tempo per generarla è un decimo di secondo, non servirà a molto. ■

BIBLIOGRAFIA

http://security.stackexchange.com/questions/53138/relationship-of-password-strength-and-key-strength
https://en.wikipedia.org/wiki/Key_stretching
https://it.wikipedia.org/wiki/Derivazione_di_una_chiave_crittografica
https://it.wikipedia.org/wiki/Principio_di_Kerckhoffs
https://it.wikipedia.org/wiki/Chiave_(crittografia)

I sorgenti dei programmi citati nell'articolo possono essere scarivati da qui:
https://github.com/nannib/crypto