

Crittografia e statistica

Aspetti statistici nell'ambito delle comunicazioni sicure

Davide Eynard

Indice generale

1. Introduzione.....	2
1.1. Terminologia.....	2
2. Crittografia classica.....	2
2.1. Cifratura di Cesare.....	3
2.2. Cifrature monoalfabetiche.....	5
2.2.1. XOR.....	10
2.3. Cifrature polialfabetiche.....	11
3. Crittografia moderna.....	16
3.1. Cifrature simmetriche.....	16
3.2. Cifrature asimmetriche.....	17
3.2.1. Generazione di numeri primi.....	18
3.3. Funzioni one-way.....	19
3.3.1. Riduzione dello spazio delle chiavi.....	19
3.3.2. Attacchi a dizionario.....	20
3.3.3. Birthday attack.....	21
Bibliografia.....	23
Webografia.....	23

1. Introduzione

Lo scopo di questo testo è mostrare l'importanza della scienza statistica nell'ambito delle comunicazioni sicure: la crittologia, infatti, non solo si appoggia a metodi probabilistici per l'analisi dei documenti che contengono dati nascosti, ma spesso basa tutta la propria sicurezza su conclusioni statistiche. Allo stesso tempo si cercherà di offrire un breve scorcio sul mondo delle comunicazioni sicure, tramite esempi e descrizioni di tecniche crittografiche "classiche" e moderne, insieme ad alcuni suggerimenti per cercare di aggirarle.

1.1. Terminologia

Un messaggio viene normalmente chiamato *testo in chiaro* (*plaintext*). Il processo tramite il quale si vuole manipolare un messaggio in modo da nascondere il significato viene chiamato *cifratura* (*encryption*) e il risultato di tale processo è il *testo cifrato* (*ciphertext*). Il processo inverso, quello cioè che permette di ottenere il testo in chiaro a partire dal testo cifrato, si chiama *decifratura*.

La scienza (o forse è meglio dire l'arte) che crea e usa i sistemi di cifratura si chiama *crittografia*, mentre quella che studia le tecniche per decifrare i messaggi cifrati si chiama *crittanalisi*. Quella branca della matematica che si occupa sia di crittografia che di crittanalisi viene chiamata *crittologia*.

Un *algoritmo crittografico* è la funzione matematica utilizzata per la cifratura e la decifratura: la sicurezza dell'algoritmo può basarsi unicamente sul fatto che rimanga segreto il modo in cui esso funziona, oppure appoggiarsi alla segretezza di una *chiave*. Un assioma fondamentale per la crittologia, enunciato per la prima volta da A. Kerckhoffs nel diciannovesimo secolo, afferma che la segretezza deve risiedere interamente nella chiave: se questo era vero allora lo è a maggior ragione adesso, in campo informatico, visto che si può ricostruire un algoritmo tramite *reverse-engineering* del programma che lo utilizza.

Per alcuni algoritmi la chiave di cifratura e quella di decifratura sono diverse: se l'una può essere sempre calcolata a partire dall'altra, allora l'algoritmo si dice *simmetrico*. In caso contrario, esso viene chiamato *algoritmo a chiave pubblica* (o *asimmetrico*): questo nome deriva dal fatto che solitamente la chiave utilizzata per cifrare i messaggi è pubblicamente distribuibile, mentre quella per decifrare deve rimanere segreta.

2. Crittografia classica

La crittografia definita "classica" è presente nella cultura umana da migliaia di anni: basti pensare che il primo degli algoritmi trattati in questo capitolo porta il nome di Giulio Cesare (e non è neanche il più vecchio!). Per evidenti motivi, essa è nata per la cifratura di testi, cioè insiemi di caratteri alfabetici, piuttosto che per quella di dati, cioè insiemi di byte. Tuttavia, non sarà difficile passare dall'una all'altra: sarà sufficiente considerare, anziché un alfabeto di 21 elementi (26 se utilizziamo quello inglese), uno di 256, corrispondenti a tutti i possibili valori che un byte può assumere. Gli esempi che seguono alla descrizione dei vari algoritmi di cifratura mostreranno più in dettaglio come è possibile effettuare quest'operazione.

2.1. Cifratura di Cesare

La cifratura di Cesare è uno dei più antichi, oltre che più semplici, esempi di cifratura di un testo. Esso consiste nel sostituire ogni lettera del messaggio con quella che la precede nell'alfabeto di n posizioni: il numero n può essere considerato la *chiave* dell'algoritmo di cifratura.

Ad esempio, volendo cifrare il messaggio

"Questo messaggio deve rimanere segreto"

con la chiave "12", dovremmo:

- 1) per comodità, eliminare gli spazi (che non vengono convertiti) e portare tutte le lettere al maiuscolo (visto che l'algoritmo non fa differenza fra maiuscolo e minuscolo);
- 2) modificare ogni lettera del testo in chiaro secondo la seguente corrispondenza:

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
O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N

Una volta eseguita la sostituzione, il testo cifrato sarà:

EISGHCASGGOUUWCRSJJSFWAOBSFSGSUFSHC

Per decifrare il testo, è necessario invece utilizzare un alfabeto spostato *a destra* di 12 caratteri, cioè modificare ogni lettera del testo cifrato secondo la corrispondenza *inversa* a quella precedente:

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
M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L

In questo modo si avrà nuovamente:

QUESTOMESSAGGIODEVERIMANERESGRETO

Si noti che la chiave utilizzata per cifrare è sostanzialmente la stessa utilizzata per decifrare: l'algoritmo è quindi *simmetrico* e la segretezza della chiave è indispensabile per avere qualche possibilità di mantenere il messaggio segreto. Si presume, quindi, che le persone che scambiano messaggi cifrati in questo modo si accordino preventivamente sulla chiave da utilizzare.

Naturalmente, un algoritmo di questo tipo è assolutamente banale sia da individuare sia da decifrare: il numero possibile di chiavi, infatti, si riduce al numero delle lettere

dell'alfabeto meno una (25 per l'alfabeto inglese, 20 per quello italiano), rendendo praticabile anche a mano un attacco a *forza bruta* (*brute force*), che consiste nel provare tutte le chiavi possibili.

Figura 2.1: Codice sorgente del programma *caesar.pl*, che esegue la cifratura di Cesare.

```
#!/usr/bin/perl
#
# caesar.pl
# by Davide Eynard, 2003
#
# Quick and dirty implementation of Caesar cipher: given a number "n" and
# a text, it crypts the text changing each letter with the one which comes
# "n" letters before in the alphabet.
#
# Note: the used alphabet is the english one
#
$key = $ARGV[0];
die ("Key is too big\n") if abs($key)>26;

@text = <stdin>;
$all = join('',@text);      # put all text on one line
$all =~ tr/a-z/A-Z/;       # put all letters uppercase
$all =~ tr/A-Z//cd;        # trim away everything which is not A-Z

@alltxt=split(//,$all);
for (@alltxt) {
    $char = ord($_)-$key;
    $char += 26 if $char<65;
    $char -= 26 if $char>90;
    print chr($char);
}
print "\n";
```

In Figura 2.1 è mostrato il codice Perl che applica la cifratura di Cesare a un messaggio, consentendo di specificare la chiave: è possibile inserire un valore positivo da 1 a 26 per cifrare e uno negativo per decifrare o viceversa. Ad esempio:

Figura 2.2: Esempi di esecuzione del programma *caesar.pl*

```
$ perl caesar.pl 12
Questo messaggio deve rimanere segreto
EISGHCASGGOUWCERSJSFWAOBFSFGSUFSHC

perl caesar.pl -12
EISGHCASGGOUWCERSJSFWAOBFSFGSUFSHC
QUESTOMESSAGGIODEVERIMANERESEGRETO

$ perl caesar.pl 17 <la_divina_commedia.txt>
UJMRERWJLXVVNMRJMRMJWCNJURPQRNARRWONAWXLJWCXRRWLXVRWLJUUJLXVNMJMRMJWCNJUUNPQRNARMRO
RXANWIJWNUJZDJUNCAJCCJMNUNYNWNYDWRVNWCMNERIRNMNVNARCNYANVRMNUNERACLXVRWLJRULJWCX
YARVXMNUJYARVJYJACNUJZDJUNBRLQRJVJRWONAWXWNUZDJUUJDCCXANOJYAXNVRXJDCCCJUXYNAJWNUVNI I
XMNULJVVVRWVRWBCAJERCJVRARCAJRYNADWJBUEJXBLDAJLQUJMRARCCJERJNAJBVJAARCJJQRZDJWCXJ
MRAZDJUNAJLXBJMDAJNBCJBUEJBUEJPPRJNJB...
```

2.2. Cifrature monoalfabetiche

Fu qui che Legrand, dopo aver riscaldato di nuovo la pergamena, me la sottopose. I caratteri che seguono apparivano in rosso, tracciati in modo grossolano tra il teschio e il capretto:

53\$\$&305))6*;4826)4\$);806*;48&8#60))85;1\$(:;\$*8&83(88)5*&;46(;88*96*?;8)*\$(;485(;
5*&2:*\$(\$;4956*2(5*4)8#8*;4069285);)6&8)4\$;\$;1(\$9;48081;8:\$1;48&85;4)485&528806*8
1(\$9;48;(88;4(\$?34;48)4\$;161;:188?\$?;

- E con ciò -dissi, restituendogli la pergamena- non mi sembra per questo più chiaro. Se per conquistare tutti i tesori di Golconda dovessi risolvere questo enigma, sono certo che non riuscirei a guadagnarmeli.

- Eppure -disse Legrand- la soluzione è più facile di quanto non si pensi al primo superficiale sguardo.

E. A. Poe: "Lo scarabeo d'oro"

Le cifrature monoalfabetiche, pur avendo alcune somiglianze con la cifratura di Cesare, comportano una (relativamente) maggiore difficoltà dal punto di vista crittanalitico e sono decisamente più interessanti dal punto di vista statistico. Anche in questo caso, l'algoritmo non è dei più complessi: semplicemente, ad ogni carattere si decide di farne corrispondere un altro (e uno solo), questa volta non "traslando" l'alfabeto ma mescolandone le lettere.

In questo caso, le corrispondenze possibili non sono più 25 ma un numero notevolmente più alto: esse sono tante quante le possibili permutazioni dei caratteri dell'alfabeto. È possibile descrivere in modo operativo una di queste permutazioni come segue:

- 1) scelgo una delle 26 lettere dell'alfabeto da far corrispondere alla lettera A;
- 2) scelgo una delle rimanenti 25 lettere da far corrispondere alla lettera B;
- 3) scelgo una delle rimanenti 24 lettere da far corrispondere alla lettera C;
- ...
- 25) scelgo una delle rimanenti 2 lettere da far corrispondere alla lettera Y;
- 26) scelgo l'ultima lettera, da far corrispondere alla lettera Z.

Il numero totale di permutazioni è, quindi, $26 \cdot 25 \cdot 24 \cdot \dots \cdot 1$, cioè $26!$, corrispondente a un ordine di grandezza di 10^{26} : questo significa che, con un computer in grado di provare un milione di permutazioni al secondo, non saremmo comunque in grado di trovare quella giusta prima della fine dell'universo! Inoltre, per rendere più complicato il lavoro del crittanalista si possono utilizzare per l'alfabeto segreto dei caratteri non alfabetici.

Per fortuna, la statistica ci viene in aiuto rendendo la decifrazione del messaggio quasi banale: come descritto anche all'interno de "Lo scarabeo d'oro" di Poe, è sufficiente fare un'analisi della frequenza con cui compaiono le varie lettere cifrate all'interno del testo. Infatti, queste avranno sempre la medesima frequenza delle loro controparti in chiaro, rendendo possibile in un primo tempo una sostituzione dei caratteri che compaiono più frequentemente di tutti, quindi la deduzione di intere parole a partire dalle lettere che sono già state decifrate. Naturalmente, questo tipo di analisi è tanto migliore quanto più lungo è il testo da decifrare.

Ma qual è la frequenza delle lettere dell'alfabeto all'interno di un normale messaggio in chiaro? Per saperlo è necessario consultare una *tabella di frequenze* relativa alla lingua in cui si suppone sia stato scritto il messaggio: infatti, da un idioma all'altro può cambiare la frequenza con cui vengono utilizzate le varie lettere. Per quanto riguarda l'inglese, è possibile trovare diverse tabelle di frequenze su Internet: ad esempio, sul sito <http://www.data-compression.com/english.html> sono addirittura disponibili statistiche del primo, del secondo e del terzo ordine (per motivi di spazio, in Figura 2.3 appare solo la tabella di frequenze del prim'ordine). Per l'italiano, invece, si è deciso di scrivere un breve programma in Perl per contare le frequenze e ricavare da queste una tabella.

Figura 2.3: Tabella delle frequenze per la lingua inglese.

Inglese			Italiano		
			Numero di lettere: 4331428		
Lettera	Frequenza		Lettera	Totale	Frequenza
A	0.0651738		a	485647	0.1121217
B	0.0124248		b	37088	0.0085625
C	0.0217339		c	206544	0.0476850
D	0.0349835		d	164868	0.0380632
E	0.1041442		e	526145	0.1214715
F	0.0197881		f	52715	0.0121704
G	0.0158610		g	88101	0.0203399
H	0.0492888		h	62996	0.0145439
I	0.0558094		i	423419	0.0977551
J	0.0009033		j	36	0.0000083
K	0.0050529		k	1	0.0000002
L	0.0331490		l	263064	0.0607338
M	0.0202124		m	114564	0.0264495
N	0.0564513		n	308441	0.0712100
O	0.0596302		o	410971	0.0948812
P	0.0137645		p	119679	0.0276304
Q	0.0008606		q	33383	0.0077072
R	0.0497563		r	287860	0.0664585
S	0.0515760		s	238177	0.0549881
T	0.0729357		t	250733	0.0578869
U	0.0225134		u	141806	0.0327389
V	0.0082903		v	92244	0.0212964
W	0.0171272		w	3	0.0000007
X	0.0013692		x	374	0.0000863
Y	0.0145984		y	10	0.0000023
Z	0.0007836		z	22559	0.0052082

Il codice sorgente del programma Perl è mostrato in Figura 2.4: per ottenere le statistiche mostrate in questa pagina sono stati dati in pasto al programma il *Decameron*, la *Gerusalemme liberata*, *I promessi sposi*, *La Divina commedia*, *l'Orlando furioso* e il *Principe*, tutti scaricati dal sito <http://www.liberliber.it/>. Come si può notare dai dati in figura il numero totale di lettere supera i quattro milioni, sufficienti per rendere la statistica abbastanza affidabile.

Figura 2.4: Il codice sorgente del programma countletters.pl.

```
#!/usr/bin/perl
#
# countletters.pl
# by Davide Eynard, 2003
#
# Counts letters inside a text file and prints statistics (number, frequency)
# about them. Also saves these stats in a data file for later retrieval and
# incremental statistics.
#
# NOTE: This still doesn't take into account àèìòù and other accents.
#

my %letter; # %letter is a hash, which can be described as an array where
            # indexes can be any kind of variable type
my $total = 0; # initialize "total letters" counter
my $IOC = 0; # Index Of Coincidence

my $DATAFILE = "data.txt"; # data file name. Format:
                            # Letters: <total letters>
                            # letter (tab) count (tab) frequency

# open data file if it exists
my $in = open (IN, "<$DATAFILE");
if ($in){
    print "Found $DATAFILE: reading previous stats...\n";
    # retrieve info about previous stats
    while (<IN>){
        if (/^letters:\s+(\d+)/i){ $total=$1; } # total letters
        else{
            chomp;
            my ($let,$cnt,$per) = split ("\t",$_);
            #printf "$let\t$cnt\t%.07f\n",$per;
            # update per-letter stats:
            $letter{$let}{count}=$cnt;
        }
    }
}
close IN;

# now read lines from stdin and count letters in them
while (< >) {
    for (/[a-zA-Z]/g){
        # I use just the lowercase [lc()] value of the letter as a hash key,
        # so I can count uppercase and lowercase letters together.
        $letter{lc($_)}{count}++;
        $total++; # this is the total number of letters
    }
}

foreach $k (keys %letter){
    # I calculate the percentages only now so I have to do max = 26
    # (maximum number of different alphabetic letters found) divisions
    $letter{$k}{percent}=$letter{$k}{count}/$total;
    $IOC +=($letter{$k}{count}*( $letter{$k}{count}-1 ))/($total*( $total-1 ));
}

```

```
# note that now the hash has this format:
# letter -> "letter name" -> "count" = number of times the letter was found
# letter -> "letter name" -> "percent" = frequency of the letter
open (OUT, ">$DATAFILE");
print "\nStatistics based on $total letters:\n";
print OUT "Letters: $total\n";
print "-----\n";
for (sort keys %letter){
    printf "$_\t$letter{$_}{count}\t%.07f\n", $letter{$_}{percent};
    print OUT "$_\t$letter{$_}{count}\t$letter{$_}{percent}\n";
}
print "-----\n";
printf "Index Of Coincidence: %.03f\n\n", $IOC;
close OUT;
```

A questo punto, non ci resta altro da fare che provare i dati su un testo cifrato. La figura seguente mostra un testo preso dal libro *Codici e segreti*, di Simon Singh (ed. Rizzoli).

Figura 2.5: Un testo cifrato con cifratura monoalfabetica.

```
BLQVCDDBRMLXTJX MQQMVDXVC NXGGX NBJM NB LMTC NUCLC X
RB LBRXVC M RAVBDXVX NBXJVC MG AMTNXGMFVC RCQVM GM
AMGAX NXGGM QMVXJX NXG QMGMOOC VXMGX X BG VX DXNXDM
BG QMGLC NXGGM LMTC APX RAVBDXDM. MGGCVM BG VX LUJC BG
ACGCVX NXGGM IMAABM X B RUCB QXTRBXVB GC JUVFMVCTC; GX
WBUTJUVX NXB RUCB IBMTAPB RB RABCGRXVC X B RUCB WBT-
CAAPB FMJJXDMTC GUTC ACTJVC GMGJVC. BG VX CVNBTC ACT
MUJCVBJM NB IMV DXTBVX B LMWPB, B AMGAXB X WGB
MRJVCGCWB. BG VX NBAPBMVC MB RMQBXTJB NB FMFBGCTBM:
"APBUTKUX GXWWXVM KUXRJM RAVBJJUV M X LB IMVM ACT-
CRAXVX GM RUM BTJXVQVXJMOBCTX BTNCRXVM GM QCVQCV M,
WGB RB LXJJXVM UTM ACGGMTM NCVC BTJCVTC MG ACGGC X
RMVM JXVOC TXG WCDXVTC NXG VXWTC."
MGGCVM MAACVRXVC JUJJB B RMQBXTJB NXG VX, LM TCT VBU-
RABVCTC M GXWWXVX GM RAVBJJUV M X M NMVTX GBTJXVQVXJ-
MOBCTX MG VX. ACRB BG VX FMGNMRRMV VBLMRX LCGJC
JUVFMJC, GM RUM IMAABM AMLFBC ACGCVX X B RUCB
NBWBJMVB IUVTCT MJJXVVBJB. GM VXWBTM, RACRRM NMGGX
QMVCX NXG VX X NXB RUCB NBWBJMVB, XTJVC TXGGM RMGM
NXG ACTDBJC . GM VXWBTM QVXR M NBVX: "C VX, DBDB BT
XJXVTC. TCT JB JUVFBTC B JUCB QXTRBXVB X BG ACGCVX NXGGM
JUM IMAABM TCT AMLFB. AX UT UCLC TXG JUC VXWTC APX
QCRBXXNX GC RQBVBXC NXWGB NXB RMTJB. IBT NMB JXLQB NB
JUC QMNVX IUVTCT JVCDMJB BT GUB BTJXGGXJJC, BTJXGGBWTOM
X RMQBXTOM RBLBGX MGGM RMQBXTOM NXWGB NXB. RBAAPX
BG VX TMFUACNCTCRCV JUC QMNVX GC QCRX M AMQC NXB
LMWPB, NXWGB BTNCDBTB, NXB AMGAXB X NXWGB MRJVCGCWB.
QCBAPX UTC RQBVBXC RUQXVBCVX, RABXTOM X BTJXGGBWXTOM,
GBTJXVQVXJMOBCTX NXB RCWTB, GM ACTCRAXTOM NXWGB
XTBWL, GM RCGUOBCTX NXGGX ACRX NBIIBABGB RCTC RJMJX
JVCDMJB BT GUB, NMTBXGX, M AUB BG VX QCRX TCLX FMGJ-
MOOMV; CVM RBM APBMLMJC NMTBXGX XN XWGB BTNBAPXVM
GBTJXVQVXJMOBCTX". GM QVBLM QMVCGM APBMDX X CJPXGGC
```

L'analisi delle frequenze sul testo preso in esame, prodotta utilizzando lo script `Perl countletters.pl`, è la seguente:

Figura 2.6: Risultato dell'analisi statistica sul testo cifrato.

Statistics based on 1361 letters:

a	55	0.0404115
b	168	0.1234386
c	131	0.0962528
d	18	0.0132256
f	12	0.0088170
g	111	0.0815577
i	11	0.0080823
j	73	0.0536370
k	2	0.0014695
l	24	0.0176341
m	145	0.1065393
n	55	0.0404115
o	16	0.0117561
p	14	0.0102866
q	34	0.0249816
r	62	0.0455547
t	87	0.0639236
u	40	0.0293902
v	106	0.0778839
w	29	0.0213079
x	168	0.1234386

Index Of Coincidence: 0.078

Cominciamo subito col notare che le lettere che ricorrono più volte sono la *b* e la *x*, entrambe con la stessa frequenza, seguite da *m* e *c*. Senza troppi dubbi, possiamo stabilire che queste corrisponderanno alle lettere in chiaro *a* e *i* *o*: resta solo da decidere in che ordine. Per trovare la giusta corrispondenza, iniziamo associando convenzionalmente alla lettera cifrata *x* la traduzione *e* (infatti, *x* appare da sola già nella prima riga, facendo pensare alla lettera *e* usata come congiunzione) e a *c* la traduzione *o* (guardando la tabella in Figura 2.3 possiamo affermare che *c* resta comunque, fra le quattro, la lettera meno utilizzata). Potremmo anche mettere in corrispondenza, seguendo le statistiche, *b* con *a* e *m* con *i*, ma a una successiva analisi ci accorgeremmo che, in questo caso particolare, tale soluzione sarebbe errata (d'altra parte, la statistica ci fornisce delle probabilità, non delle certezze).

A questo punto, poiché ci sono delle parole di tre lettere che finiscono per *e*, possiamo supporre che esse siano dei *che* cifrati, trovando così altre due corrispondenze: (*a*,*c*), (*p*,*h*). Ora la parola, ancora parzialmente cifrata, *MccoVReVo* ci offre un grande aiuto: ci suggerisce che ad *m* è associata la lettera *a* (e quindi a *b* la vocale *i*) e inoltre ci "regala" le coppie (*v*,*r*), (*r*,*s*). Dopo questa sostituzione sarà possibile, utilizzando la medesima tecnica di completamento delle parole, decifrare in modo molto semplice il resto del messaggio: in Figura 2.7 compare il testo decifrato a cui sono stati aggiunti accenti e apostrofi, i quali erano stati omessi per rendere più complicata la decifrazione.

Figura 2.7: Il testo decifrato.

improvvisamente apparvero delle dita di mano d'uomo e si misero a scrivere dietro al candelabro sopra la calce della parete del palazzo reale e il re vedeva il palmo della mano che scriveva. allora il re muto' il colore della faccia e i suoi pensieri lo turbarono; le giunture dei suoi fianchi si sciolsero e i suoi ginocchi battevano l'uno contro l'altro. il re ordino' con autorita' di far venire i maghi, i caldei e gli astrologi. il re dichiaro' ai sapienti di babilonia:

"chiunque leggerà questa scrittura e mi farà conoscere la sua interpretazione indossera' la porpora, gli si mettera' una collana d'oro intorno al collo e sara' terzo nel governo del regno."

allora accorsero tutti i sapienti del re, ma non riuscirono a leggere la scrittura e a darne l'interpretazione al re. così il re baldassar rimase molto turbato, la sua faccia cambio' colore e i suoi dignitari furono atterriti. la regina, scossa dalle parole del re e dei suoi dignitari, entro' nella sala del convito. la regina prese a dire: "o re, vivi in eterno. non ti turbino i tuoi pensieri e il colore della tua faccia non cambi. c'è un uomo nel tuo regno che possiede lo spirito degli dei santi. fin dai tempi di tuo padre furono trovati in lui intelletto, intelligenza e sapienza simile alla sapienza degli dei. sicché il re nabucodonosor tuo padre lo pose a capo dei maghi, degli indovini, dei caldei e degli astrologi. poiché uno spirito superiore, scienza e intelligenza, l'interpretazione dei sogni, la conoscenza degli enigmi, la soluzione delle cose difficili sono state trovate in lui, danièle, a cui il re pose nome baltazzar; ora sia chiamato danièle ed egli indicherà l'interpretazione". la prima parola chiave è othello

2.2.1. XOR

Uno dei comandi più utilizzati per la cifratura in ambito informatico è l'operatore binario XOR: esso viene descritto in questo contesto perché può essere usato anche per eseguire una cifratura monoalfabetica, lavorando sui byte anziché sulle lettere. XOR opera nel seguente modo:

A\B	1	0
1	0	1
0	1	0

Questo significa che, per ogni coppia A,B di bit in ingresso, XOR (A,B) restituisce 1 solo se uno (e uno solo) dei due ha il valore 1, 0 altrimenti (questo è il motivo per cui esso viene anche chiamato OR esclusivo). Lo XOR fra due byte viene eseguito

ripetendo l'operazione di XOR per tutte le coppie di bit che li compongono. Ad esempio:

A = 117 dec (01110101 bin)

B = 73 dec (01001001 bin)

C = A xor B = 00111100 bin = 60 dec

A = C xor B = 01110101 bin

Come si può notare anche dalla seconda delle due operazioni, una delle proprietà principali di XOR è la seguente:

$$\text{se } A \text{ XOR } B = C, \text{ allora } C \text{ XOR } B = A$$

cioè utilizzando B come chiave è possibile sia cifrare A ottenendo C, sia decifrare C ottenendo A: la simmetria (in termini crittografici) di questo operatore è uno dei motivi principali per cui esso è così largamente utilizzato. Inoltre, è semplice notare che, usando una chiave di un byte, ogni valore viene modificato tramite una cifratura monoalfabetica (anche se lo spazio delle chiavi è limitato a soli 255 elementi, escludendo il caso banale dello 0).

2.3. Cifrature polialfabetiche

I sistemi di cifratura polialfabetica costituiscono ancora un passo avanti, sia a livello di sicurezza sia dal punto di vista dell'interesse statistico, rispetto ai sistemi monoalfabetici. Essi infatti consentono di utilizzare diverse variazioni dell'alfabeto di cifratura a seconda della lettera che si desidera nascondere, rendendo quindi inutile il calcolo delle frequenze utilizzato per la cifratura monoalfabetica.

Un esempio classico è quello di Vigenère: la sua *chiffre indéchiffable* sfruttava 26 variazioni della cifratura di Cesare da usare alternativamente, una per ogni lettera della parola usata come chiave di cifratura. La procedura descritta in dettaglio è la seguente:

- 1) Si sceglie una chiave e la si ripete tante volte quante servono ad avere lo stesso numero di lettere all'interno del testo in chiaro: ad esempio, se il testo è NELMEZZODELCAMMINDINOSTRAVITA e la chiave è (forse un po' banale) DANTE, prepareremo le due stringhe

NELMEZZODELCAMMINDINOSTRAVITA

DANTEDANTEDANTEDANTEDANTEDANT

- 2) Si cifra ogni lettera del testo in chiaro utilizzando l'alfabeto individuato dalla corrispondente lettera della parola chiave: ad esempio, da N e D otteniamo la lettera che si trova alla colonna N e alla riga D (Q), da E ed A otteniamo E, da L e N otteniamo Y e così via.

NELMEZZODELCAMMINDINOSTRAVITA

DANTEDANTEDANTEDANTEDANTEDANT

QEYFICZBWIOCNFQLNQBRRSKEYIGT

Figura 2.8: La tavola di Vigenère.

	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
A	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
B	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	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	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

In versione moderna, si può applicare una cifratura polialfabetica ai byte anziché alle lettere dell'alfabeto utilizzando l'operatore XOR e una chiave di più caratteri: già con soli 8 byte di chiave (64 bit) un algoritmo di *brute force* dovrebbe provare 2^{64} chiavi (eventualmente riducibili, conoscendo il sottoinsieme di byte -ad esempio solo caratteri alfanumerici- da cui è stata creata la chiave). Se ne deduce che un algoritmo polialfabetico, classico o moderno che sia, rappresenta comunque un bel passo avanti rispetto al suo predecessore e, almeno apparentemente, non è risolvibile con la medesima tecnica.

Nella frase precedente, "*almeno apparentemente*" dev'essere interpretato in questo modo: *a meno che non si sappia la lunghezza della chiave*. In quel caso, infatti, sarebbe possibile utilizzare il calcolo delle frequenze su tutte le lettere cifrate con lo stesso alfabeto (cioè, data la lunghezza n della chiave, su una lettera ogni n).

Per risolvere questo problema, William Friedman (1891-1969) inventò un test, chiamato *test di Friedman* (o *test kappa*), pubblicato nel 1920 all'interno dell'articolo "The Index of Coincidence and its Applications in Cryptography". Esso non solo è in grado di stabilire se una cifratura è monoalfabetica o polialfabetica, ma può anche aiutare il crittanalista a individuare la lunghezza della chiave.

La definizione dell'indice di coincidenza è la seguente:

Siano :

N = numero totale di lettere nel testo

n_1 = numero di A

n_2 = numero di B

...

n_{26} = numero di Z¹

Poichè

$$P(A \cap B) = P(A)P(B|A) = P(B)P(A|B)$$

e se A e B sono indipendenti :

$$P(A|B) = P(A)$$

$$P(B|A) = P(B)$$

Allora

$$P(\text{due volte } A) = P(A_1 A_2) \quad (\text{con } A_2 \text{ dipendente da } A_1)$$

$$\Rightarrow P(A_1 A_2) = P(A_1)P(A_2|A_1) = \frac{n_1}{N} \cdot \frac{n_1 - 1}{N - 1} = \frac{n_1(n_1 - 1)}{N(N - 1)}$$

Per calcolare la probabilità di trovare due lettere identiche, sommiamo tutte e 26 le probabilità :

$$IC = \sum_{i=1}^{26} \frac{n_i(n_i - 1)}{N(N - 1)}$$

Note :

1. Si considerano anche le lettere JKWXY (l'alfabeto italiano ha solo 21 lettere).

Utilizzando il programma `countletters.pl`, è possibile calcolare anche l'indice di coincidenza per una particolare lingua: i calcoli effettuati sui medesimi file di testo descritti in precedenza hanno fornito un indice di 0.073 per l'italiano. Ne segue che eseguendo gli stessi calcoli su un testo crittato con cifratura monoalfabetica dovremo ottenere lo stesso valore (in quanto, pur cambiando i simboli, la probabilità di trovarne due identici resta la stessa). Se, invece, il valore è molto più basso (ad esempio 0.050) l'algoritmo sarà probabilmente polialfabetico.

Figura 2.9: Alcuni indici di coincidenza per lingue diverse.

Arabic	0.075889	French	0.074604	Japanese	0.077236	Serbo Cr.	0.064363
Danish	0.070731	German	0.076667	Malay	0.085286	Spanish	0.076613
Dutch	0.079805	Greek	0.069165	Norwegian	0.069428	Swedish	0.064489
English	0.066895	Hebrew	0.076844	Portuguese	0.074528	Random	0.038461
Finnish	0.073796	Italian	0.073294	Russian	0.056074		

Una volta stabilito il tipo di cifratura utilizzato, e ammesso che questa sia polialfabetica, è necessario scoprire la lunghezza della chiave. Per risolvere questo problema è necessario calcolare la probabilità di coincidenze ogni due lettere, ogni tre, ogni quattro e così via, finché non si trova un indice di coincidenza verosimile. Se questo accade, ad esempio, all' n -esima lettera allora possiamo affermare che la lunghezza della chiave è n . Nel caso di una cifratura di Vigenère, poi, scoprire il messaggio sarà banale: il calcolo delle frequenze, eseguito una lettera ogni n , permetterà di trovare facilmente gli n caratteri della chiave. Come esempio, utilizziamo ancora un testo dal libro *Codici e segreti*:

Figura 2.10: Un testo cifrato con la tecnica di Vigenère.

```
KQOWEFVJPUJUUNUKGLMEKJINMWUXFQMKJBGWRLFNFGHUDWUUMBSVLP SNCMUEKQCTESWREEKOYSSIWCTUAXYO
TAPXPLWPNTCGOJBGFQHTDWXIZAYGFFNSXCSEYNCTSSPNTUJNYTGGWZGRWUUNEJUUEAPYMEKQHUIDUXFPUGUY
TSMTFFSHNUOCZGMRUWEYTRGKMEEDCTVRECFBDJQCUSWVBNLGOYLSKMTFVJJTWWMFMWPNMEMTMHRSPXFSSK
FFSTNUOCZGMDOEYOYEKCPJRGPMURSKHFRSEIUEVGOYCWIXIZAYGOSAANYDOEYJLWUNHAMEBFELXYVLWNOJNS
IOFRWUCCESWKWIDGMUCGOCRUWGNMAAFFVNSIUDEKQHCEUCPFCMPVSUDGAVEMNYMAMVLFMAOYFNTQCUAFVVFJN
XKLNEIWCWODCCULWRIFTWGMUSWOVMATNYBUHTCOCWFYTNMGYTQMKBBNLGFBTWOJFTWGNTEJKNEEDCLDHWTVB
UVGFBIJGYIIDGMVRDGMPLSWGJLAGOEKJOFKKNYNOLRIVRVUHEIWUURWGMUTJCDNBKGMIBDGMEEYGUOTDGG
QEUJYOTVGGBRUJYS
```

L'indice di coincidenza calcolato da `countletters.pl` è 0.044, quindi la cifratura è molto probabilmente polialfabetica. Per calcolare la lunghezza della password il testo è stato dato in pasto al programma `Perl checkioc.pl`, che effettua il calcolo degli indici su diversi "sfasamenti" del testo e ne restituisce i valori offrendone anche una rappresentazione grafica.

Figura 2.11: Il codice sorgente del programma `checkioc.pl`.

```
#!/usr/bin/perl
#
# checkioc.pl
# by Davide Eynard, 2003
#
# Given a polyalphabetically ciphered text file, it checks the Index
# of Coincidence for various possible lengths of the password.

my $MAXLEN = "8"; # maximum password length: check until this size

@text=<stdin>;

$all=join('',@text) ; # put all text on one line
$all =~ tr/a-z/A-Z/ ; # put all letters uppercase
$all =~ tr/A-Z//cd ; # trim away everything which is not a letter A-Z
@alltxt=split(//,$all) ;
for ($skip=1;$skip<=$MAXLEN;$skip++){
    my $count = 0;
    $header=substr($all,0,$skip) ;
    $shifted = substr($all,$skip).$header ; # creates a shifted text
    @shifttxt=split(//,$shifted) ;
    foreach $i(0..$#alltxt){
        if($alltxt[$i] eq $shifttxt[$i]) { $count++ ;}
    }
    my $index = $count/$#alltxt;
    printf("IoC for password length=$skip is: %2f\t",$index);
    for ($j=.01;$j<$index;$j+=.01){print " ";}
    print "\n";
}
```


L'output del programma eseguito sul testo cifrato di Figura 2.10 è il seguente:

Figura 2.12: Output del programma checkioc.pl.

```
$ perl checkioc.pl <vigenere.txt
IoC for password length=1 is: 0.048093 ****
IoC for password length=2 is: 0.023217 **
IoC for password length=3 is: 0.034826 ***
IoC for password length=4 is: 0.044776 ****
IoC for password length=5 is: 0.096186 *****
IoC for password length=6 is: 0.054726 *****
IoC for password length=7 is: 0.024876 **
IoC for password length=8 is: 0.024876 **
```

La conclusione è che la chiave molto probabilmente sarà di 5 caratteri: basterà inserire questo valore nel programma `countletters_poly.pl` (una variazione di `countletters`) per ottenere la password e avere così la possibilità di decifrare il testo.

Figura 2.12: Il codice sorgente del programma countletters_poly.pl.

```
#!/usr/bin/perl
#
# countletters_poly.pl
# by Davide Eynard, 2003
#
# Counts letters inside a polyalphabetically ciphered text file, given the
# password length, and tries to find the right password given that "e" is
# the most frequent letter.

my $MOSTFREQ = 'e';
my $OFFSET = ord($MOSTFREQ)-ord('a');
my $PASSLEN = 5; # password length
my @letter;
my $i = 0; # simple counter

# read lines from stdin and count letters in them
while (<)& {
    for (/[a-zA-Z]/g){
        $letter[$i]{lc($_)}{count}++;
        $i++; $i=0 if ($i==$PASSLEN);
    }
}

# walk the array of hashes to find the most frequent letters
for ($i=0;$i<$PASSLEN;$i++){
    my $MAX;my $LET;
    foreach $k (sort keys %{$letter[$i]}){
        if ($MAX < $letter[$i]{$k}{count}){
            $MAX = $letter[$i]{$k}{count};
            $LET = chr(ord($k)-$OFFSET);
        }
    }
    print $LET;
}
print "\n";
```

3. Crittografia moderna

Già dopo aver letto l'ultimo esempio, relativo alla cifratura di Vigenère (che, fra quelle polialfabetiche, è forse una delle più semplici da aggirare), si può intuire l'importanza che ha la chiave rispetto all'algoritmo, una volta che quest'ultimo sia scelto in modo appropriato. Si pensi infatti a una cifratura polialfabetica con una chiave lunga tanto quanto il testo da nascondere: sarebbe impossibile in pratica trovare il testo in chiaro tramite il calcolo dell'indice di coincidenza! Questa tecnica, chiamata del *blocco monouso*, è tuttora giudicata impossibile da aggirare e, anche se non è molto pratica, viene ancora utilizzata per proteggere transazioni sicure su Internet.

Nella crittografia moderna, in generale, gli algoritmi sono tutti ben noti: la loro sicurezza si basa sulla segretezza della chiave e su proprietà matematiche che dimostrano l'impossibilità pratica di trovarla se non in tempi astronomici. Per lo stesso motivo, saranno assenti prove sperimentali di crittanalisi a favore di una descrizione teorica, con particolare attenzione alle considerazioni statistiche relative agli algoritmi più utilizzati.

3.1. Cifrature simmetriche

Al giorno d'oggi, algoritmi simmetrici come il DES si appoggiano a operazioni che creano *confusione* (modificando l'aspetto dei dati) e *diffusione* (modificando la posizione dei dati all'interno di un file) per rendere impossibili in pratica le analisi statistiche, tanto efficienti nello studio degli algoritmi classici. Si tenga inoltre presente il fatto che, lavorando con generici dati e non più con semplici testi, può essere molto più complicato (se non assolutamente inutile) utilizzare un'analisi delle frequenze o un indice di coincidenza. Anche conoscendo parte del testo in chiaro non è possibile decifrare il resto del messaggio se non provando un *brute force* sulla chiave: ma quali sono i tempi necessari per eseguire quest'operazione?

Figura 3.1: stime di tempo per un attacco brute force hardware nel 1995.

Average Time Estimates for a Hardware Brute-Force Attack in 1995						
Cost	LENGTH OF KEY IN BITS					
	40	56	64	80	112	128
\$100 K	2 seconds	35 hours	1 year	70,000 years	10^{14} years	10^{19} years
\$1 M	.2 seconds	3.5 hours	37 days	7000 years	10^{13} years	10^{18} years
\$10 M	.02 seconds	21 minutes	4 days	700 years	10^{12} years	10^{17} years
\$100 M	2 milliseconds	2 minutes	9 hours	70 years	10^{11} years	10^{16} years
\$1 G	.2 milliseconds	13 seconds	1 hour	7 years	10^{10} years	10^{15} years
\$10 G	.02 milliseconds	1 second	5.4 minutes	245 days	10^9 years	10^{14} years
\$100 G	2 microseconds	.1 second	32 seconds	24 days	10^8 years	10^{13} years
\$1 T	.2 microseconds	.01 second	3 seconds	2.4 days	10^7 years	10^{12} years
\$10 T	.02 microseconds	1 millisecond	.3 second	6 hours	10^6 years	10^{11} years

Come accennato in precedenza parlando dell'operatore XOR, il numero di chiavi possibili per una lunghezza di n bit è 2^n e già una chiave di 8 byte (64 bit) può richiedere dei tempi astronomici per essere trovata. Inoltre, poiché ogni bit in più fa raddoppiare il numero di tentativi necessari per trovare la chiave, è sufficiente

aggiungere un byte (un solo carattere!) per rendere l'attacco 256 volte più difficile. La Figura 3.1 mostra alcune stime di tempo (relative al 1995) per un attacco *brute force* con hardware dedicato.

3.2. Cifrature asimmetriche

Le cifrature asimmetriche utilizzano funzioni matematiche in grado di creare una *chiave pubblica* a partire dalla quale sia praticamente impossibile trovare la corrispondente *chiave privata*. In particolare, RSA utilizza il seguente procedimento per la generazione delle chiavi:

- 1) vengono scelti due numeri primi p e q , entrambi molto grandi, che dovranno essere tenuti segreti;
- 2) viene calcolato $N=pq$ e viene scelto un numero a caso e tale che e stesso, $(p-1)$ e $(q-1)$ siano primi fra loro: la coppia (N,e) costituisce la *chiave pubblica*;
- 3) applicando l'*algoritmo di Euclide*, viene calcolato un intero d tale che

$$e \cdot d \equiv 1 \pmod{(p-1)(q-1)}$$

- 4) il numero d costituisce la *chiave privata*: per cifrare un numero x , sarà sufficiente calcolare $C=x^e \pmod{N}$; per decifrarlo, invece, si avrà $D=C^d \pmod{N}$.

Per fare un esempio molto semplice, supponiamo di aver scelto i due numeri primi $p=17$ e $q=11$ e il numero casuale $e=7$. Allora $N=187$ e $d=23$. Supponendo di voler cifrare la lettera X, corrispondente al codice ASCII 88, calcoleremmo:

$$C=88^7 \pmod{187}=11$$

Per decifrare la lettera C, invece, è necessario calcolare

$$D=11^{23} \pmod{187}=88$$

ottenendo così la lettera in chiaro.

Già da questo semplice esempio è facile capire la complessità, a livello matematico ma anche computazionale, dell'intero algoritmo: non solo le funzioni esponenziali in aritmetica dei moduli sono "a senso unico" (vedremo questo concetto più in dettaglio in seguito) e non consentono quindi l'inversione del processo di cifratura, ma i numeri primi con cui si ha a che fare hanno in realtà dimensioni tali da impedire, per limiti fisici dei computer attuali, la fattorizzazione del numero N . Basti pensare che le chiavi utilizzate per l'esempio hanno una dimensione di pochi bit, mentre è abbastanza comune trovare in circolazione chiavi di 2048 bit.

In Figura 3.2 sono mostrati gli anni MIPS necessari a fattorizzare numeri dai 512 ai 2048 bit (un anno MIPS corrisponde a un anno di lavoro di una macchina da un MIPS: per farsi un'idea, un Pentium 100 è una macchina da circa 50 mips). Si noti anche che lo *Special Number Field Sieve* era l'algoritmo più rapido che ci fosse nel 1995 (anno a cui risalgono i dati) per eseguire la scomposizione in fattori primi.

Figura 3.2: Calcolo degli anni MIPS necessari per la fattorizzazione delle chiavi da 512 a 2048 bit.

Factoring Using the Special Number Field Sieve	
# of bits	Mips-years required to factor
512	<200
768	100,000
1024	$3 \cdot 10^7$
1280	$3 \cdot 10^9$
1536	$2 \cdot 10^{11}$
2048	$4 \cdot 10^{14}$

3.2.1 Generazione di numeri primi

L'operazione che, nell'ambito della cifratura a chiave pubblica, ha probabilmente maggior interesse statistico è la generazione dei numeri primi che costituiscono la chiave privata. Essi devono essere molto grandi in modo che sia impossibile in pratica fattorizzarne il prodotto, ma com'è possibile essere certi che essi siano primi senza poterne calcolare i fattori?

Il *piccolo teorema di Fermat* ci viene in aiuto, dicendo che se m è primo, allora per ogni numero intero a non multiplo di m

$$a^{m-1} \equiv 1 \pmod{m}$$

Naturalmente, provare gli infiniti a non è meno dispendioso che cercare di fattorizzare un numero gigantesco. Esistono, tuttavia, degli algoritmi in grado di provare statisticamente la primalità: questo significa che non si avrà mai la certezza che i numeri considerati siano primi, ma si potrà rendere piccola a piacere la probabilità di errore. L'algoritmo più utilizzato è quello di *Rabin e Miller* e sfrutta il piccolo teorema di Fermat:

- 1) si scelga innanzitutto un numero casuale p su cui effettuare il test di primalità. Si calcoli inoltre b , tale che sia il numero di volte che 2 divide $p-1$ (cioè, tale che 2^b sia la più grande potenza di 2 che divide $p-1$). Infine, si calcoli m , tale che $p=1+2^b \cdot m$;
- 2) si scelga un numero casuale a inferiore a p ;
- 3) sia $j=0$ e $z=a^m \pmod{p}$;
- 4) se $z=1$ o se $z=p-1$, allora p passa il test e può essere primo;
- 5) se $j>0$ e $z=1$, allora p non è primo;

- 6) sia $j=j+1$. Se $j < b$ e $z \neq p-1$, sia $z=z^2 \bmod p$ e si torni al passo precedente. Se invece $z=p-1$, allora p passa il test e può essere primo;
- 7) se $j=b$ e $z \neq p-1$, allora p non è primo.

Uno dei motivi principali per cui questo algoritmo resta il più utilizzato è dato dal fatto che la probabilità che la previsione sia errata è molto bassa per ogni singolo passo: infatti, nell'interpretazione più pessimistica dell'algoritmo, tre quarti dei valori restituiti come validi lo sono effettivamente. Questo significa che iterando la procedura per t volte, avremo una probabilità di errore che decresce come $1/4^t$. All'atto pratico l'algoritmo è addirittura molto più efficiente, restituendo valori corretti circa il 99,9% delle volte, abbassando la probabilità di errore per un n da 256 bit a $1/2^{51}$.

3.3. Funzioni one-way

Le funzioni "a senso unico" (*one way*) sono funzioni piuttosto semplici da calcolare, ma decisamente più complicate da invertire, cioè data x è semplice ottenere $f(x)$, ma è praticamente impossibile (principalmente per motivi computazionali) da $f(x)$ ottenere x . Grazie a questa loro proprietà, sono una componente fondamentale per buona parte dei protocolli e degli algoritmi utilizzati dalla crittografia moderna.

Un caso particolare di funzioni *one way* è costituito dalle funzioni *hash*, che convertono una stringa di byte di lunghezza variabile in una di lunghezza fissa (chiamata *valore hash* o *hash*). Esse sono utilizzate in quantità massiccia a livello informatico, ad esempio per conservare le password degli utenti nei sistemi Unix, per creare *session id* nei siti web, per verificare la validità di codici seriali relativi a protezioni software e così via. Poiché una funzione hash ha dimensione finita, è banale dedurre che la corrispondenza fra gli elementi del dominio e quelli del codominio sia di molti a uno: esistono cioè diverse coppie di messaggi (M, M') tali che $H(M)=H(M')$. Per questo motivo un algoritmo hash è buono non solo se risponde a piccole variazioni nella chiave con grandi variazioni nell'hash, ma anche se è *collision-free*, cioè ha una bassissima probabilità che due chiavi generino lo stesso hash.

3.3.1. Riduzione dello spazio delle chiavi

Anche a causa del loro principale utilizzo (quello di cifrare password degli utenti), lo spazio delle chiavi cifrate con delle funzioni hash può in certi casi essere notevolmente ridotto: basti pensare al fatto che la tastiera stessa non mette a disposizione (se non attraverso l'inserimento di codici tramite tastierino numerico) tutti e 256 i valori rappresentabili da un byte. A questo bisogna aggiungere il fatto che gli utenti sono soliti scegliere password semplici da digitare, se non addirittura normali parole o brevi sequenze di numeri. La figura seguente mostra come lo spazio delle chiavi possa essere drasticamente ridotto se si tengono in conto queste limitazioni.

Figura 3.3: Numero delle chiavi possibili per le varie riduzioni dello spazio delle chiavi.

Number of Possible Keys of Various Keyspaces					
	4-Byte	5-Byte	6-Byte	7-Byte	8-Byte
Lowercase letters (26):	460,000	$1.2 \cdot 10^7$	$3.1 \cdot 10^8$	$8.0 \cdot 10^9$	$2.1 \cdot 10^{11}$
Lowercase letters and digits (36):	1,700,000	$6.0 \cdot 10^7$	$2.2 \cdot 10^9$	$7.8 \cdot 10^{10}$	$2.8 \cdot 10^{12}$
Alphanumeric characters (62):	$1.5 \cdot 10^7$	$9.2 \cdot 10^8$	$5.7 \cdot 10^{10}$	$3.5 \cdot 10^{12}$	$2.2 \cdot 10^{14}$
Printable characters (95):	$8.1 \cdot 10^7$	$7.7 \cdot 10^9$	$7.4 \cdot 10^{11}$	$7.0 \cdot 10^{13}$	$6.6 \cdot 10^{15}$
ASCII characters (128):	$2.7 \cdot 10^8$	$3.4 \cdot 10^{10}$	$4.4 \cdot 10^{12}$	$5.6 \cdot 10^{14}$	$7.2 \cdot 10^{16}$
8-bit ASCII characters (256):	$4.3 \cdot 10^9$	$1.1 \cdot 10^{12}$	$2.8 \cdot 10^{14}$	$7.2 \cdot 10^{16}$	$1.8 \cdot 10^{19}$

3.3.2. Attacchi a dizionario

Uno degli attacchi più comuni ai sistemi di cifratura che fanno uso di funzioni hash è quello che utilizza un *dizionario* di parole: poiché non è possibile invertire la funzione hash, si cifra allora un lungo elenco di possibili password sperando che alcune di queste corrispondano a quelle scelte dagli utenti. Su Internet sono presenti diversi dizionari già pronti, che contengono le password più comunemente utilizzate, e programmi che le testano applicandovi alcune variazioni e aumentando così la probabilità di trovarne alcune valide. Anche se, in generale, gli attacchi di questo tipo fanno maggior uso di tecniche di social engineering piuttosto che di analisi statistiche, la percentuale di successi è incredibilmente alta: attorno al 40%!

Per ridurre in parte l'efficacia di questo tipo di attacchi alle password viene solitamente aggiunto un *salt*, un valore di 2 byte che a parità di parola chiave applica 4096 possibili modifiche aggiuntive. Questo preclude la possibilità di cifrare l'intero dizionario una volta per tutte all'inizio del procedimento: visto che solitamente il salt dipende dal nome utente, sarà necessario operare su tutte le parole del dizionario per ogni password che si desidera scoprire. Ad ogni modo, per avere una discreta sicurezza, quando si sceglie una password è bene tener presenti i seguenti suggerimenti:

- 1) non scegliere password corrispondenti al proprio nome, a quello della propria moglie o dei figli, né tantomeno date di nascita, anniversari, matrimoni e così via;
- 2) non scegliere parole comuni né citazioni da film (*pencil*, essendo sia una parola comune che una citazione dal film *Wargames*, è altamente sconsigliata... lo stesso vale per *Joshua*, perché è anche un nome di persona);
- 3) aggiungere la propria data di nascita in fondo al nickname *non* costituisce una buona password;
- 4) utilizzare una grande varietà di caratteri alfanumerici e non: una password costituita da lettere maiuscole e minuscole, numeri e segni di interpunzione è molto più difficile da indovinare;
- 5) per creare password complicate è possibile appoggiarsi a *passphrase*: ad esempio, dalla frase "*Let's do the Timewarp again!*" si può ricavare la password "*LdtTa!*", altrimenti difficile da ricordare.

3.3.3. Birthday attack

Gli attacchi *brute force* che si possono attuare contro una funzione a senso unico sono di due tipi diversi. Il primo è quello classico: dato un hash $H(M)$, si cerca di trovare un M' tale che $H(M)=H(M')$. Nel secondo tipo di attacco, invece, si cerca di generare una *collisione*, trovando due messaggi casuali, M e M' , tali che $H(M)=H(M')$. In quest'ultimo caso, le probabilità di riuscita sono molto più alte: la dimostrazione spiegherà anche perchè questo viene chiamato *birthday attack*.

Calcoliamo la probabilità che, su n persone, una compia gli anni il nostro stesso giorno:

Sia

$A = \text{qualcuno compie gli anni nel giorno } x$
 $\Rightarrow A^C = \text{nessuno compie gli anni nel giorno } x$

Allora :

Eventi possibili $= 365 \cdot 365 \cdot \dots \cdot 365 (n \text{ volte}) = 365^n$
 Eventi favorevoli per $A^C = 364 \cdot 364 \cdot \dots \cdot 364 (n \text{ volte}) = 364^n$

Si conclude che :

$$P(A) = P(1 - A^C) = 1 - \left(\frac{364}{365}\right)^n$$

Come su può notare eseguendo alcuni semplici calcoli, per avere una probabilità maggiore di 0.5 che almeno una persona compia gli anni il nostro stesso giorno è necessario che n sia maggiore o uguale a 253. Se, invece, desideriamo che all'interno di un gruppo di n persone almeno due abbiano lo stesso compleanno (in un giorno qualsiasi), la probabilità è la seguente:

Sia

$A = \text{almeno due persone compiono gli anni lo stesso giorno}$
 $\Rightarrow A^C = \text{nessuno compie gli anni lo stesso giorno}$

Allora :

Eventi possibili $= 365 \cdot 365 \cdot \dots \cdot 365 (n \text{ volte}) = 365^n$
 Eventi favorevoli per $A^C = 365 \cdot 364 \cdot \dots \cdot (365 - (n - 1)) = (365)_n$

Si conclude che :

$$P(A) = P(1 - A^C) = 1 - \frac{(365)_n}{365^n}$$

Incredibilmente, per avere una probabilità $P(A)$ maggiore del 50% è sufficiente avere un n maggiore o uguale a $23!$ Allo stesso modo, se per trovare un messaggio che corrisponda a un hash dato è necessario cifrare 2^m messaggi, trovarne due che abbiano lo stesso hash richiede solo $2^{m/2}$ tentativi casuali: questo significa che un computer in grado di trovare, in 600000 anni, un messaggio che corrisponda a un particolare hash sarebbe in grado di trovarne una coppia in grado di generare un altro hash in circa un'ora.

Bibliografia

Bruce Schneier, *Applied Cryptography: Protocols, Algorithms and Source Code in C*, second edition

Edgar Allan Poe, *Lo scarabeo d'oro*

Simon Singh, *Codici e segreti*, ed Rizzoli 1999

Webografia

Marco Campedelli, *La genesi della crittografia: le prime scritture criptate*

<http://www.rcvr.org/varie/pgp/storia.htm>

Didier Müller, *Le test de friedman*

<http://www.jura.ch/lcp/cours/dm/codage/stat/friedman.html>

James J. Gillogly, *Cyphertext only cryptanalysis of Enigma*

<http://members.fortunecity.com/jpeschel/gillog1.htm>

Nam Phamdo, *Statistical Distributions of English Text*

<http://www.data-compression.com/english.html>

Liber Liber

<http://www.liberliber.it/>