Appunti di Crittografia

Luca Battistin

Dicembre 2020

Sommario

Queste note sono rivolte agli studenti del quinto anno della specializzazione di informatica presso l'Istituto Tecnico Industriale Marzotto-Luzzati di Valdagno. Si tratta di appunti raccolti e aggiornati nel corso di diversi anni allo scopo di offrire un percorso organico attraverso i concetti e i sistemi crittografici, approfondendo le basi algoritmico-matematiche e fornendo qualche riferimento storico. Si descrivono i principali algoritmi e i loro impieghi. Si distinguono gli algoritmi a chiave simmetrica da quelli a chiave pubblica, con particolare riferimento all'RSA. Si introducono le funzioni di Hash, quindi i sistemi ibridi come la firma digitale, l'https, il sistema PGP, TOR e le monete digitali come il Bitcoin. Si accenna ad alcuni tipi di attacco. Gli algoritmi presentati hanno scopo didattico quindi trascurano l'efficienza per favorire la comprensione di quanto accade.

Nota: Si consiglia di NON stampare questo scritto perché ancora in versione β e in continuo aggiornamento. Si ringrazia caldamente chiunque voglia segnalare errori o imprecisioni.

Indice

1 Introduzione

Il termine crittografia (o criptografia) deriva dall'unione di due parole greche: **kryptòs** ($\kappa\rho\upsilon\pi\tau o$), "occulto", e **graphos** ($\gamma\rho\alpha\varphi\omega\sigma$), "scrittura". Esso indica il processo di rendere un messaggio illeggibile alle persone non autorizzate a riceverlo. La crittografia va vista nella cornice della **sicurezza informatica** che definiamo come:

sicurezza informatica: l'insieme di misure software, hardware, strutturali, procedurali atte a garantire la riservatezza, l'integrità e la disponibilità dei dati[1].

La terminologia di questo ramo della scienza informatica non è difficile ma vanno precisate le definizioni di alcuni termini su cui facilmente si crea confusione.

Crittografia : è l'arte di camuffare, rendere inintelligibile, un messaggio a chiunque non abbia la chiave per decriptarlo.

Crittoanalisi : è l'arte di infrangere i sistemi crittografici

Crittologia : è lo studio dei sistemi con cui inviare messaggi riservati su mezzi insicuri. Si può considerare come l'unione di crittografia e crittanalisi

testo in chiaro : (o plaintext) è il messaggio intelligibile che il mittente vuole far pervenire al destinatario autorizzato ma non agli altri soggetti (eventualmente avversari o spioni) che venissero in possesso del messaggio lungo il canale di comunicazione

testo cifrato : (o ciphertext) è il messaggio reso inintelligibile dal metodo di criptazione

All'interno dei sistemi crittografici si distinguono i codici e i cifrari:

il Codice è l'associazione tra intere parole del messaggio in chiaro e intere parole del messaggio cifrato.

il Cifrario è la corrispondenza tra i singoli simboli del messaggio in chiaro e di quello cifrato

[1] Questa definizione si rifà allo standard ISO 27002 secondo il quale la sicurezza dell'informazione è caratterizzata dalla *CIA triad: Confidentiality, Integrity, Availability.*

si può dire che il *codice* lavora sul Dizionario mentre il *cifrario* lavora sull'alfabeto. La distinzione tra i due non è però così netta perché molti cifrari lavorano su gruppi (blocchi) di simboli.
Infine va resa esplicita la definizione di

Steganografia : è l'arte di nascondere l'esistenza di un messaggio all'interno di una comunicazione pubblica che apparentemente trasporta tutt'altra informazione. Deriva dalle parole greche *steganos* ($\sigma\tau\varepsilon\gamma\alpha\nu o\zeta$) "coperto" e *graphos* ($\gamma\rho\alpha\varphi\omega\sigma$) "scrittura".

Ad esempio si può nascondere del testo all'interno di un'immagine digitale usando il bit meno significativo del codice RGB. Ne vedremo una applicazione nelle prossime note. Per ora si accenna al tool *steghide* di linux da usare in combinazione con l'editor di immagini Gimp. Nell'antichità è famosa[2] la storia raccontata da Erodoto a proposito di un certo Isteo che, volendo mandare dalla corte persiana un messaggio al genero, il tiranno Aristagora di Mileto, fece rapare a zero un fedele schiavo, gli fece tatuare il messaggio sul cuoio capelluto, aspettò che i capelli ricrescessero e poi lo mandò al tiranno con l'istruzione di farsi rapare nuovamente.

2 Schema crittografico

Tutte le nostre considerazioni faranno riferimento allo schema di Figura 1 dove *Alice* è il mittente di un messaggio che deve rimanere segreto mentre attraversa un canale insicuro e *Bob* è il destinatario. Nell'attraversare il canale insicuro il messaggio può essere letto da *Eve*, sempre in ascolto per carpire i segreti di *Alice* e *Bob*. L'algoritmo di cifratura usato da *Alice* ha come input il testo in chiaro (*plaintext*) e la chiave di cifratura k_C e come output il messaggio cifrato (*ciphertext*) mentre *Bob* usa un algoritmo di decifratura i cui input sono il messaggio criptato e la chiave k_D.

3 Cifrari a Sostituzione

3.1 Cesare

Cominciamo a studiare uno tra i più semplici e famosi cifrari a sostituzione: quello che Giulio Cesare sembra usasse per comunicare con le proprie legioni. si tratta di *sostituire* ogni simbolo del messaggio in chiaro con quello che lo segue di k posizioni nell'alfabeto. Se, ad esempio $k = 3$ allora il messaggio

[2] l'ha resa tale David Khan nel suo *The codebreakers*

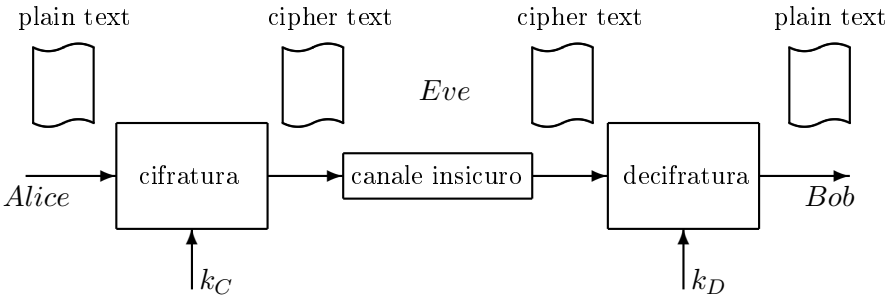


Figura 1: schema crittografico

```
DIVIDE ET IMPERA
```

diventa

```
GMZMGH HY MPSHVD
```

Va notato che a quel tempo le lettere dell'alfabeto Latino erano 23, esclusivamente maiuscole, per la precisione: A B C D E F G H I K L M N O P Q R S T V X Y Z. Quindi Cesare scriveva in codice le sue missive riservate, sostituendo ciascuna lettera con quella che la segue di tre posti nell'alfabeto, ordinato ciclicamente: la A con la D, la B con la E, la C con la F, la D con la G, ... la V con la Z, la X con la A, la Y con la B e la Z con la C.

Se indichiamo con a la posizione nell'alfabeto del simbolo in chiaro (partendo da zero) e con c quella del simbolo cifrato possiamo scrivere la formula di cifratura:

$$c = a + k \pmod{n} \tag{1}$$

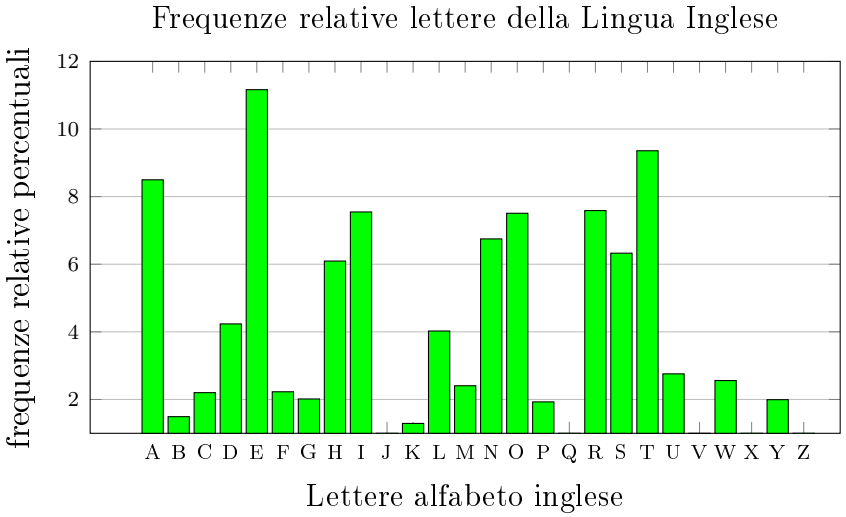
Dove n è la cardinalità dell'alfabeto (23 per Giulio Cesare, 26 per l'alfabeto inglese) e k è la costante di spostamento (*shift*).

La formula di decifrazione deve eseguire uno spostamento all'indietro, quindi sarà semplicemente $a = c - k \pmod{n}$. Uno strumento che implementa con facilità le operazioni appena descritte è il disco cifrante di Leon Battista Alberti[3]: si tratta di due dischi concentrici, rotanti uno rispetto all'altro e contenenti le lettere dell'alfabeto. Basta ruotare il disco esterno del numero di posizioni scelto come chiave per ottenere la corrispondenza tra lettere del testo in chiaro e quelle del testo cifrato. Un attacco esaustivo a forza bruta in questo caso (ovvero quando la sostituzione si ottiene da una semplice spostamento di tutte le lettere dell'alfabeto a destra) deve controllare solo al massimo 25 valori, ma questo algoritmo ci introduce all'algebra

[3]Leon Battista Alberti (Genova, 14 febbraio 1404 – Roma, 25 aprile 1472) è stato un architetto, scrittore, matematico, umanista, crittografo, linguista, filosofo, musicista e archeologo italiano; fu una delle figure artistiche più poliedriche del Rinascimento. [wikipedia]

modulare, molto usata nella crittografia moderna.

In generale, tuttavia, i cifrari a sostituzione possono prevedere una qualsiasi delle permutazioni[4] di 26 simboli e il numero totale di tali permutazioni è enorme: $26! = 4,03 \cdot 10^{26}$. Se poi si considera la possibilità di usare simboli diversi dalle lettere dell'alfabeto, il numero di possibili cifrari sembra tendere all'infinito e per tanto tale metodo fu ritenuto per secoli indecifrabile. Si pensi che se un milione di persone lavorassero in parallelo provando una combinazione al secondo, sarebbe necessario un tempo pari a circa 1000 volte l'età dell'universo per provarle tutte! Fu il matematico arabo **Al-Kindi**[5] il primo a trovare il modo di rompere qualsiasi cifrario a sostituzione notando che la frequenza relativa delle lettere in un qualsiasi testo di una lingua naturale tende a stabilizzarsi su un valore preciso. Nel testo *Sulla decrittazione della corrispondenza cifrata* egli spiegò che contando la frequenza dei simboli del testo cifrato e confrontando tali frequenze relative con quelle tipiche[6] della lingua in cui di ipotizza sia scritto il testo in chiaro si può associare ad ogni simbolo del testo cifrato il corrispondente simbolo del testo in chiaro.



[4]Il calcolo combinatorio, che studia i raggruppamenti che si possono ottenere con un numero n di oggetti disposti in un numero k di posti, indica col nome di *Permutazione* (senza ripetizione) ogni raggruppamento in cui i posti sono tanti quanti gli oggetti, ovvero n; Le *Disposizioni* sono invece i raggruppamenti in cui i posti sono diversi (minori) del numero di oggetti e conta l'ordine; se l'ordine non conta, siamo in presenza di *Combinazioni*.

[5]Abu Yusuf Yaqub ibn Ishaq al-Sabbah Al-Kindi fu un filosofo, astronomo, fisico e matematico arabo del nono secolo D.C. E' considerato uno dei più grandi dei filosofi islamici dei suoi tempi.

[6]Al-Kindi non disponeva ovviamente delle tavole delle frequenze relative delle lingue naturali, che ancora non esistevano, ma suggeriva di prendere un testo in chiaro abbastanza lungo e di contare le frequenze delle lettere incontrate.

L'arte della crittoanalisi richiede intuito, perseveranza e molte altre tecniche che sfruttano, ad esempio, la forte ridondanza di qualsiasi lingua naturale, ma l'analisi delle frequenze è sicuramente il grimaldello primo che permette di aprire qualsiasi cifrario a sostituzione. Esempi notevoli di crittoanalisi dei cifrari a sostituzione sono offerti dalla letteratura : Nel racconto "Lo scarabeo d'oro" (The Gold-Bug) di E.A. Poe[7] l'autore guida il lettore nella decifrazione di una misteriosa pergamena, mentre ne "The dancing men" A.Conan Doyle spiega come Shelock Holmes riesce a decifrare dei bigliettini riportanti degli omini danzanti che l'assassino inviava alla sua vittima.

3.2 Affine

Una formula di cifratura un po' più complicata è quella che va sotto il nome di *cifrario Affine* dove oltre allo scostamento c'è anche un fattore di proporzionalità:

$$c = \alpha \cdot a + \beta \pmod{n} \tag{2}$$

La formula per decifrare il codice affine si trova invertendo l'equazione lineare 2 per cui si trova

$$a = (c - \beta) \cdot \frac{1}{\alpha} \pmod{n}$$

Ma cosa significa $\frac{1}{\alpha} \pmod{n}$? I numeri razionali non son ammessi. Quale carattere è rappresentato da $\frac{1}{3}$ o da $\frac{1}{9}$. Il problema è trovare il *moltiplicativo inverso in algebra modulare* α^{-1}. Per ora ci basta notare che l'inverso moltiplicativo di $\alpha = 9$ modulo 26 è $\alpha^{-1} = 3$ perché

$$9 \cdot 3 \equiv 1 \pmod{26}$$

. Va notato che α non può essere un numero qualsiasi ma deve essere primo rispetto ad n. Questo concetto matematico è centrale anche nell'algoritmo RSA quindi lo tratteremo a fondo nel prossimo paragrafo. La formula decifrativa per il codice Affine è quindi

$$a = (c - \beta) \cdot \alpha^{-1} \pmod{n} \tag{3}$$

3.3 Vigenère

Il cifrario che prende il nome da Blaise de Vigenère[8] è un esempio di cifrario polialfabetico per cui ogni carattere del testo in chiaro viene traslato in base

[7] Edgar Allan Poe (1809 – 1849) è stato uno scrittore, poeta, critico letterario, giornalista, editore e saggista statunitense. Si era inoltre costruito fama di infallibile *code-breaker* affermando che non esisteva codice segreto che egli non potesse penetrare.

[8] Blaise de Vigenere nacque nel villaggio di Saint Pourcain tra Marsiglia e Parigi nel 1523. Nel suo libro *Traicté des ciffres*, oltre che discutere i cifrari polialfabetici, illustra

alle lettere di una parola chiave. Può essere visto come un cifrario di Cesare a n chiavi, dove n è il numero di lettere della parola segreta.
Nella tabella sono state evidenziate le righe relative alla parola chiave TRENI e a come potrebbe essere codificata la lettera G. Se ad esempio, la lettera G comparisse per qualche motivo cinque volte di seguito nel messaggio originale, verrebbe cifrata nella stringa ZXKTO. Oltre alla matrice illustrata nella Tabella 1, anche il disco di Alberti può essere facilmente usato per la cifratura e decifratura dei cifrari polialfabetici. Vedremo che esso ispirò anche la costruzione della macchina Enigma.

L'algoritmo riportato di seguito implementa i cifrari appena spiegati:

```
/* Luca Battistin - Cifrari a Sostituzione - 2013 rev gen2017 */
/* GNU GPL */
/*******************************************************
   cifrario a Sostituzione di
   cesare -- opzione -c
   affine -- opzione -a
   vigenere -- opzione -v
   la chiave è inserita dall'utente
   prende il file in chiaro come parametro
   produce il file cifrato come output
   OTTIMIZZATO per Linux x86 a 64 bit
*******************************************************/
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#include <string.h>
//funzione che calcola il simbolo cifrato
int cifratura(int a, char tipo, char* key, int pos);
int main(int argc, char** argv){
    //controllo parametri passati
    if(argc != 3){
        printf("sintassi: %s [tipocfr] nomefile\n",argv[0]);
        printf("tipocfr:\n-c (cesare)\n-a (affine)\n-v (vigenere)\n");
        exit(1);
```

il sistema a chiave autocifrante nel quale è lo stesso crittogramma a servire da chiave. Il sistema oggi universalmente chiamato Vigenère non è di fatto una sua invenzione. E' stato a lungo considerato inviolabile ma fu Chrales Babbage - l'inventore della macchina differenziale - a escogitare (nel 1800) un metodo di rottura. Purtroppo, qualcuno continuò a considerarlo inviolabile anche nel XX secolo: esso fu utilizzato, ad esempio, dall'esercito italiano nella prima guerra mondiale.)

Tabella 1: Tabella per la cifratura Vigenere

	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

```
25      }
26      //acquisisce la chiave
27      char chiave[100],k;
28      int k1,k2;
29      char tipocfr = argv[1][1];
30      if (tipocfr!='v' && tipocfr!='a' && tipocfr!='c'){
31          printf("tipocfr:\n-c (cesare)\n-a (affine)\n-v (vigenere)\n");
32          exit(1);
33      }
34      printf("inserisci la chiave\n");
35      fgets(chiave,99,stdin);
```

```
36      //apertura file da cifrare in lettura
37      FILE* fin=fopen(argv[2],"rt");
38      if(fin==NULL){
39          printf("Errore in apertura del file %s\n",argv[2]);
40          printf("sintassi: %s [tipocfr] plaintextfile\n",argv[0]);
41          exit(1);
42      }
43      //apertura file cifrato in scrittura
44      char nomeout[100];
45      strcpy(nomeout,"cifrato_");
46      strcat(nomeout,argv[2]);
47      FILE* fout=fopen(nomeout,"wt");
48      if(fout==NULL){
49          printf("Errore in apertura del file %s\n",nomeout);
50          exit(1);
51      }
52      //lettura input carattere per carattere
53      char i=0,c,a;
54      for(a = fgetc(fin); a != EOF; a = fgetc(fin))
55     {
56        //conversione in minuscolo
57          if(isupper(a))
58              a+=('a'-'A');
59          //si cifrano le sole lettere
60          if(isalpha(a)){
61              printf("carattere: %c :\t",a);
62              a = a - 'a'; //numero tra 0 e 25
63              printf("%d\t", a);
64              //formula cifratura:
65              c = cifratura(a,tipocfr,chiave,i); //
66              printf("cifrato in %d \t",c);
67              c = c + 'a'; // simbolo cifrato
68              printf("%c\n",c);
69              i++;
70          }
71          else{
72              printf("carattere %c non cifrato\n",a);
73              c = a;
74          }
75          // scrittura sul file di output
76          fputc(c,fout);
```

```
77      }
78      // CHIUSURA DEI FILE
79      fclose(fin);
80      fclose(fout);
81      printf("%d caratteri cifrati nel file: %s\n\n",i,nomeout);
82      return 0;
83  }
84
85  //funzione che calcola la formula di cifratura
86  int cifratura(int a,char tipo, char* key, int pos){
87      int cifrato;
88      int affine, cesare;
89      char vigenere[100],k;
90      switch (tipo){
91          case 'v':      //Vigenere
92              sscanf(key,"%s",vigenere);
93              k = strlen(vigenere);
94              cifrato = (a + (vigenere[pos%k]-'a'))%26;
95              break;
96          case 'a':      //Affine
97              sscanf(key,"%d %d",&affine,&cesare);
98              cifrato = (affine*a+cesare)%26;
99              break;
100         case 'c':     //Cesare
101             sscanf(key,"%d",&cesare);
102             cifrato = (a+cesare)%26;
103             break;
104         default:
105             printf("errore nel tipo di cifrario\n");
106             exit(1);
107     }
108     cifrato=(cifrato+26)%26;
109     return cifrato;
110 }
```

Va detto che il codice precedente è altamente inefficiente infatti ricalcola la chiave di cifratura per ogni carattere. Si è scelto però questa soluzione per mettere in evidenza che l'Affine e il Vigenère possono essere considerati versioni sempre più sofisticate di quello di Cesare.

4 moltiplicativo inverso

il moltiplicativo inverso di un numero α in algebra modulare è quel numero naturale α^{-1} che moltiplicato per α produce l'identità:

$$\alpha \cdot \alpha^{-1} \equiv 1 \pmod{n} \tag{4}$$

Vediamo qualche esempio numerico per capire come funzionano le cose. Prendiamo $n = 26$ e $\alpha = 9$. Si nota facilmente che in questo caso $\alpha^{-1} = 3$ infatti

$$9 \cdot 3 = 1 + 26$$

Questo però era un esempio fortunato. Pur continuando a considerare numeri piccoli, trovare l'inverso moltiplicativo di 11 mod 26 non è altrettanto immediato. Dopo alcuni tentativi si arriva a trovare $\alpha^{-1} = 19$ infatti

$$11 \cdot 19 = 1 + 26 \cdot 8$$

Si capisce allora che trovare l'inverso moltiplicativo in modulo n corrisponde a risolvere un'equazione lineare in due variabile del tipo

$$\alpha \cdot x + n \cdot y = 1 \tag{5}$$

Ci viene in aiuto un metodo che già conosciamo: l'algoritmo di Euclide per il calcolo del massimo comun divisore che abbiamo studiato nella sua elegante forma ricorsiva. Siano a e b numero naturali con $a > b$ [9]

$$MCD(a,b) = \begin{cases} a & \text{se } b = 0 \\ MCD(b, a \bmod b) & \text{altrimenti} \end{cases}$$

Per risolvere l'equazione 5 abbiamo bisogno dell' *algoritmo di Euclide esteso*, ovvero quello che considera anche la successione dei quozienti, oltre a quella dei resti. Il primo passo consiste nel dividere a per b ovvero scrivere a nella forma:

$$a = q_1 \cdot b + r_1$$

Se $r_1 = 0$ si è terminato e b è il MCD, altrimenti si continua scrivendo b nella forma:

$$b = q_2 \cdot r_1 + r_2$$

e così via fino a che il resto r_{k+1} non è zero:

$$a = q_1 \cdot b + r_1$$

[9] in caso contrario è sufficiente scambiarli

$$
\begin{aligned}
b &= q_2 \cdot r_1 + r_2 \\
r_1 &= q_3 \cdot r_2 + r_3 \\
\vdots &\ \vdots\ \vdots \\
r_{k-2} &= q_k \cdot r_{k-1} + r_k \\
r_{k-1} &= q_{k+1} \cdot r_k
\end{aligned}
$$

Si può dimostrare abbastanza facilmente che i due interi che risolvono l'equazione 5 sono dati dai termini finali delle due successioni:

$$x_0 = 0, \quad x_1 = 1, \quad \cdots x_n = -q_{n-1} \cdot x_{n-1} + x_{n-2} \tag{6}$$

$$y_0 = 1, \quad y_1 = 0, \quad \cdots y_n = -q_{n-1} \cdot y_{n-1} + y_{n-2} \tag{7}$$

Il metodo vale in generale per l'equazione

$$a \cdot y_n + b \cdot y_n = MCD(a, b)$$

e nel caso a e b siano primi tra loro ($MCD(a, b) = 1$) si riottiene l'equazione 5. Ecco l'algoritmo euclideo esteso:

```
/*******************************************
  Luca Battistin - Moltiplicativo inverso - 2017 GNU GPL
  Algoritmo di Euclide Esteso (iterativo)
  per trovare il moltiplicativo inverso
  in algebra modulare
  OTTIMIZZATO per Linux x86 a 64 bit
*******************************************/
#include <stdio.h>
//Funzione per Euclide classico ricorsivo
long MCDe(long, long, long*);
int main()
{
    long a,n;  //numeri primi tra loro: MCD(a,b)=1
    long mcd;  //massimo comun divisore
    long m=0;  //numero passi algoritmo
    long i;    //variabile di lavoro
    printf("Algoritmo di euclide Esteso\n");
    printf("Calcola il moltiplicativo inverso di a modulo n\n");
    printf("inserisci a e n: ");
    scanf("%ld %ld",&a,&n);
    if (a>n){
        i=a;
```

```
23          a=n;
24          n=i;
25      }
26      mcd = MCDe(n,a,&m);
27      if (mcd!=1){
28          printf("%ld e %ld non sono primi tra loro:",a,n);
29          printf(" il MCD vale: %ld\n",mcd);
30      }
31      printf("Euclide converge in %ld passi\n",m);
32      /* Calcolo dei quozienti*/
33      long R[m+1],Q[m+1];
34      R[0]=a; R[1]=n%a;
35      Q[0]=a; Q[1]=n/a;
36      i=1;
37      printf("quoz%ld\t:%ld\t\tresto%ld\t:%ld\n",i,Q[i],i,R[i]);
38      while(R[i]!=0){
39          i++;
40          R[i]=R[i-2] % R[i-1];
41          Q[i]=R[i-2] / R[i-1];
42          printf("quoz%ld\t:%ld\t\tresto%ld\t:%ld\n",i,Q[i],i,R[i]);
43      }
44      /* Calcolo dei coefficienti dell'equazione ax + by = MCD(a,b)*/
45      printf("coefficienti Euclide Esteso:\n");
46      long X[m+1],Y[m+1];
47      X[0]=0; X[1]=1;
48      Y[0]=1; Y[1]=0;
49      for(i=2;i<=m;i++){
50          X[i]=X[i-2] - X[i-1]*Q[i-1];
51          Y[i]=Y[i-2] - Y[i-1]*Q[i-1];
52          printf("X_%ld\t:%ld\t\tY_%ld\t:%ld\n",i,X[i],i,Y[i]);
53      }
54      printf("l'equazione ax + ny = MCD(a,b) ha come soluzione:\n");
55      printf("\t\t%ld * %ld + %ld * %ld = %ld\n",a,Y[m],n,X[m],mcd);
56      if (mcd == 1){
57          printf("inverso moltiplicativo di %ld mod %ld : %ld\n",a,n,X[m]);
58          X[m] = (X[m] + n) % n;
59          printf("ovvero: **********  %ld  **********\n", X[m]);
60      }
61      return 0;
62  }
63
```

```
64 long MCDe(long a, long b, long* step){
65     if (b == 0)
66         return a;
67     else{
68         (*step)++;
69         printf("MCD(%ld,%ld)=MCD(%ld,%ld)\n",a,b,b,a%b);
70         return MCDe(b,a%b,step);
71     }
72 }
```

che applicato al nostro caso produce questo output

```
Euclide converge in 4 passi
quoz1 :2 resto1 :4
quoz2 :2 resto2 :3
quoz3 :1 resto3 :1
quoz4 :3 resto4 :0
coefficienti Euclide Esteso:
X_2 :-2 Y_2 :1
X_3 :5 Y_3 :-2
X_4 :-7 Y_4 :3
l'equazione ax + by = MCD(a,b) ha come soluzione:
26 * 3 + 11 * -7 = 1
inverso moltiplicativo di 11 mod 26 : -7
ovvero: **********  19  **********
```

5 Trasposizione

L'altra famiglia di cifrari è rappresentata da quelli detti a *trasposizione* in quanto i simboli del messaggio in chiaro non vengono sostituiti ma scambiati di posto. Un semplice esercizio su cui tutti ci siamo cimentati almeno una volta è leggere le parole al contrario ma di solito i cifrari a trasposizione considerano il messaggio come una matrice di caratteri e scambiano righe e colonne.
Prendiamo ad esempio il messaggio che compare verso la fine del libro "Digital Fortress" di Dan Brown[10]

[10] In questo techno-thriller del 1998 sono contenuti molti concetti della sicurezza informatica e della crittografia, purtroppo spesso conditi con errori anche grossolani. Ciò non toglie il merito di portare il lettore dentro uno scenario plausibile e criticamente attuale: la stretta relazione tra crittografia, agenzie investigative governative, diritto alla privacy e sicurezza nazionale. Dopo tutto in questi ultimi tempi si fa sempre più spesso uso del termine *cyberwar*

PFEESESNRETMPFHAIRWEOOIGMEENNRMAENETSHASDCNSIIAAIEERBRNKFBLELODI

Ciò che salta subito agli occhi di un criptoanalista è la lunghezza del messaggio: 64, un quadrato perfetto. Se solo lo si divide in frammenti di 8 simboli

```
P F E E S E S N
R E T M P F H A
I R W E O O I G
M E E N N R M A
E N E T S H A S
D C N S I I A A
I E E R B R N K
F B L E L O D I
```

E lo si legge per colonne (separando le parole man mano che le si riconosce) si trova il messaggio in chiaro:

PRIME DIFFERENCE BETWEEN ELEMENTS RESPONSIBLE FOR HIROSHIMA AND NAGASAKI

Un algoritmo C che implementa questo tipo di trasposizione è il seguente:

```c
/* Luca Battistin - Cifrario a Trasposizione - 2013 */
/* GNU GPL */
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#include <string.h>
// considera blocchi di NXN simboli
// prende il file in chiaro come parametro del main

int main(int argc, char** argv){
//controllo parametro passato
    if(argc != 2){
        printf("sintassi: %s nomefileplaintext\n",argv[0]);
        exit(1);
    }
    int N;
    printf("inserisci N (la lunghezza sarà NXN): ");
```

```
18      scanf("%d",&N);
19      // apro in lettura il file di input
20      FILE* fin=fopen(argv[1],"rt");
21      if(fin==NULL){
22          printf("Errore in apertura del file %s\n",argv[1]);
23          exit(1);
24      }
25      //apro il scrittura il file di output
26      char nomeout[100];
27      strcpy(nomeout,"trasposto_");
28      strcat(nomeout,argv[1]);
29
30      FILE* fout=fopen(nomeout,"wt");
31      if(fout==NULL){
32          printf("Errore in apertura del file %s\n",nomeout);
33          exit(1);
34      }
35      //leggo carattere per carattere
36      char i=0,j;
37      char riga[N*N+1],a;
38      for(a = fgetc(fin); a != EOF; a = fgetc(fin))
39      {
40          if(a != '\n')
41              riga[i++]=a;
42          if(i == N*N){
43              for(i=0;i<N;i++)
44                  for(j=0;j<N;j++)
45                      fprintf(fout,"%c",riga[j*N+i]);
46              i=0;
47          }
48      }
49      //completamento ultima riga
50      if(i>0){
51          for(j=i;j<N*N;j++)
52              riga[j]=(j%26)+'a';
53          for(i=0;i<N;i++)
54              for(j=0;j<N;j++)
55                  fprintf(fout,"%c",riga[j*N+i]);
56      }
57      // CHIUSURA DEI FILE
58      fclose(fin);
```

```
59      fclose(fout);
60      printf("\ntesto trasposto nel file: %s\n\n",nomeout);
61      return 0;
62  }
```

6 La macchina Enigma

Le vicende legate alla macchina Enigma e alla sua crittanalisi rappresentano l'anello di congiunzione tra la crittografia antica e quella moderna. Il suo inventore, Arthur Scherbius, estese il concetto del disco cifrante di Alberti, mettendone tre in serie e collegandoli con un intricato cablaggio elettrico. Uno schema delle parti essenziali della macchina Enigma è rappresentato in figura 2. dove, oltre ai tre dischi, detti rotori, si nota un pannello permutatore o **plugboard** che aumenta notevolmente la complessità della cifratura scambiando coppie di lettere tra loro in base alla configurazione di 10 cavetti. Ecco cosa succede: quando l'operatore preme una lettera sulla tastiera (ad esempio la lettera H) il segnale elettrico raggiunge il pannello permutatore dove, in base ai collegamenti impostati dall'operatore, viene scambiata (nell'esempio in figura, con la lettera V); da qui il segnale elettrico raggiunge il primo rotore, che si è spostato in avanti di una posizione, dove avviene una sostituzione in base al cablaggio interno del rotore stesso. Il segnale uscente dal primo rotore diventa l'input del secondo rotore e così succede anche per il terzo, dopo di che il segnale raggiunge il riflettore, dove viene riflesso e percorre il circuito al contrario, passando per altre tre sostituzioni, prima di arrivare al pannello permutatore e finalmente accendere una lampadina indicante la lettera cifrata.

Le Lettere evidenziate dalla lampadina costituivano quindi il messaggio cifrato che veniva inviato via radio sotto forma di codice morse. Per la simmetria evidenziata dallo schema, se l'operatore ricevente impostava la propria macchina Enigma nello stesso identico modo della macchina cifrante e inseriva il messaggio cifrato, vedeva comparire sulle lampadine il testo in chiaro.
Lo spazio delle chiavi, ovvero il numero di differenti impostazioni iniziali della macchina dipende quindi da tre fattori:

orientazione dei rotori Ognuno dei tre rotori va impostato in una delle 26 lettere dell'alfabeto. Ci possono essere quindi

$$26 \times 26 \times 26 = 17.576$$

diverse impostazioni dei rotori

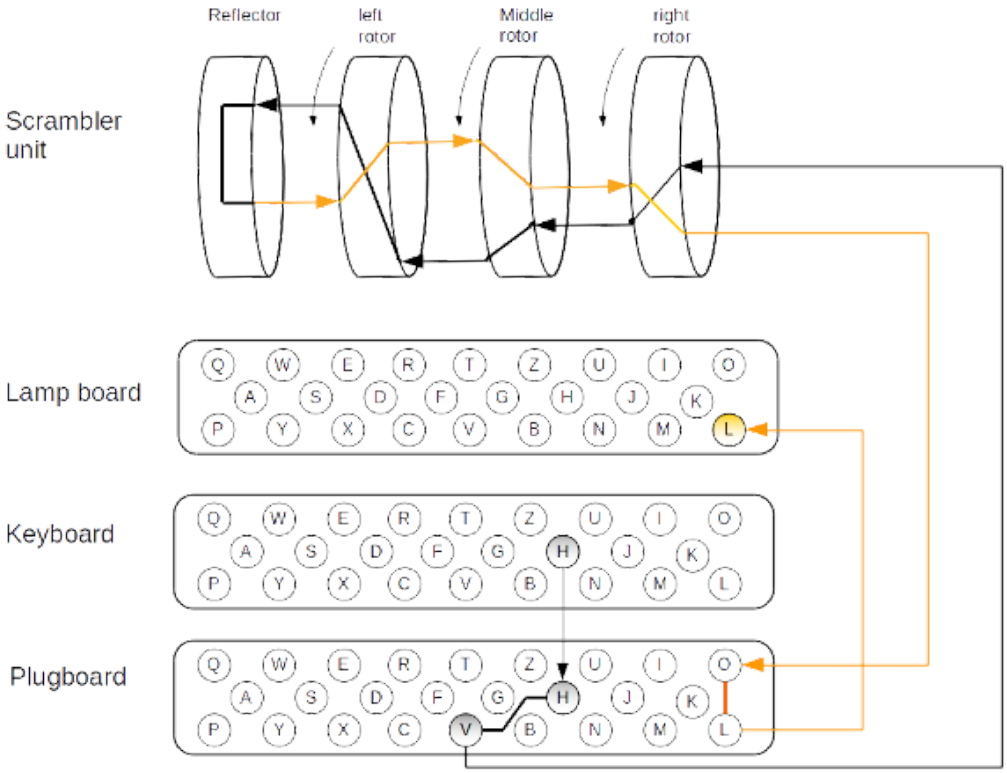


Figura 2: schema macchina Enigma

ordine dei rotori L'operatore aveva cinque diversi rotori a disposizione tra cui sceglierne tre e ordinarli. Di tratta quindi della disposizione di tre oggetti presi da un insieme di 5:

$$\frac{5!}{(5-3)!} = 5 \cdot 4 \cdot 3 = 60$$

configurazione della plugboard I modi in cui si possono collegare 26 lettere a due a due con 10 cavi si ottiene da:

$$\frac{26!}{6! \cdot 2^{10} \cdot 10!} = 150.738.274.937.250$$

Lo spazio delle chiavi è il prodotto dei tre numeri appena calcolati:

$$17576 \cdot 60 \cdot 150.738.274.937.250 = 158.962.555.217.826.360.000$$

ovvero più di 150 milioni di milioni di differenti impostazioni iniziali possibili. Questo numero esorbitante fu ciò che fece considerare la macchina un sistema impenetrabile e tale assunto fu anche parte della sua fine. Certo, per vincere contro un sistema così formidabile furono necessari lo sforzo titanico di molte persone e il genio di Alan Turing[11]. La straordinaria avventura della crittoanalisi della Macchina Enigma va oltre gli obiettivi di questo scritto ma è sicuramente una storia da conoscere[12] perché è fatta di spie, di atti eroici, di scienza, di intuizioni ed errori che sono stati determinanti per la fine della seconda guerra mondiale. In appendice si può trovare una breve cronologia degli eventi legati alla macchina Enigma. In questa sede ribadiamo però il ruolo centrale di A. Turing nel passaggio dalla crittografia antica a quella moderna oltre a ricordare la sua macchina teorica[13]e il test per l' intelligenza artificiale, pietre miliari della scienza informatica.

[11]Alan Mathison Turing (Londra, 23 giugno 1912 – Manchester, 7 giugno 1954) è stato un matematico, logico, crittografo e filosofo britannico, considerato uno dei padri dell'informatica e uno dei più grandi matematici del XX secolo. [wikipedia]

[12]Per Approfondire la storia della macchina Enigma e della sua decodifica si veda l'ottimo libro si S. Singh [2] oppure quello di Andrew Hodges: Alan Turing. The Enigma [3]

[13]La Macchina Universale di Turing è descritta nel celeberrimo articolo "On Computable Numbers". La MUT è un dispositivo teorico di cui egli si servì per dimostrare l'esistenza di assunti matematici indecidibili, all'interno dell'allora fervente dibattito sui fondamenti della matematica iniziato da David Hilbert. La MUT non va confusa con le *Bombe* di Turing: dispositivi elettromeccanici ideati per cercare la combinazione corretta dei rotori della macchina Enigma tra tutte le combinazioni possibili. Erano armadi pesanti circa una tonnellata, costituiti da tre batterie, ciascuna contenente dodici colonne di tre rotori (ovvero una macchina enigma). La fila superiore di rotori di ciascuna batteria girava ad una velocità di 120 giri al minuto.

7 Claude Shannon

Oltre alla figura A. Turing, prima di passare alla crittografia moderna, è doveroso citare Claude E. Shannon[14], padre della teoria dell'informazione. Nel celebre articolo : *A Mathematical Theory of Communication* del 1948[15] egli getta le basi teoriche della comunicazione digitale, definendo la misura della quantità di informazione e fornendo gli strumenti matematici per misurare quanta informazione può passare attraverso un canale rumoroso. Partendo dall'idea che l'informazione sia qualcosa che riduce l'incertezza e che la quantità di informazione portata da un simbolo (o messaggio) sia tanto maggiore quanto meno probabile è il simbolo stesso[16], Shannon quantifica l'informazione I_x associata all'emissione del messaggio x con la seguente formula:

$$I_x = \log_2 \frac{1}{p_x}[bit]$$

dove p_x è la probabilità associata al simbolo $x \in \mathcal{A}$, ovvero la probabilità che la sorgente S, tra tutti i simboli del proprio alfabeto $\mathcal{A}$, emetta il simbolo x.La formula diventa più chiara nel caso in cui la sorgente sia binaria (ovvero può emettere solo due simboli) e i due simboli siano equiprobabili. In tal caso ognuno di essi porta un bit di informazione, riconciliando il significato di bit come unità di misura della quantità di informazione con quello di cifra binaria. Ma se i due messaggi, che chiamiamo "0" (zero) e "1" (uno) non sono equiprobabili, l'uno porterà più di un bit di informazione, mentre l'altro porterà meno di un bit di informazione. Pensiamo ad esempio ad un allarme antincendio: il suo azionamento ("1") è, si spera, molto infrequente e porta molta informazione mentre il fatto che sia silenzioso ("0") non fa pensare "Beh, oggi non è andata a fuoco la scuola!". Se l'allarme suona mediamente due volte l'anno (per le prove di evacuazione) la quantità di informazione associata al simbolo "1" sarà:

[14] Claude Elwood Shannon (1916 - 2001) è stato un matematico ed un ingegnere. Lavorò presso i Bell laboratories dove girava per i corridoi su un monociclo e rivestiva un ruolo molto particolare: quello di accademico-ingegnere-filosofo che gli permise di produrre idee di grande ampiezza e audacia [3] e di fare esperimenti in diversi campi. Oltre che alla teoria dell'informazione, si dedicò alle macchine pensanti... Queste affinità con A. Turing li fecero subito avvicinare quando quest'ultimo visitò i Bells labs nel 1943; Shannon amava la giocoleria e recentemente è stato realizzato un docu-film sulla sua biografia dal titolo **The bit player**

[15] http://people.math.harvard.edu/∼ctm/home/text/others/shannon/entropy/entropy.pdf

[16] In altre parole l'informazione è tanto maggiore quante sono le alternative che permette di scartare. Ciò non è in disaccordo con il senso comune per cui "fa notizia" ciò che è più improbabile o meno frequente

$$I_1 = \log_2 \frac{1}{p_1} = \log_2 \frac{365}{2} = 7,5\, bit$$

Più ancora dell'informazione portata dai singoli messaggi è interessante misurare l'informazione media della sorgente, ovvero l'informazione media contenuta in tutti i suoi possibili messaggi, ovvero ciò che Shannon, nel paragrafo "CHOICE, UNCERTAINTY AND ENTROPY", definì l' Entropia H della sorgente. Se la sorgente S può emettere n simboli diversi, ognuno dei quali ha probabilità p_i di essere emesso, l'entropia di S si ottiene dalla seguente formula.

$$H = \sum_{i=1}^{n} p_i \cdot \log_2 \frac{1}{p_i}$$

Egli dimostrò che l'Entropia è massima (e quindi è massimo il suo contenuto informativo) quando i simboli sono equiprobabili e non correlati: la probabilità di un simbolo è $\frac{1}{n}$ indipendentemente dai simboli che l'hanno preceduto.

Proviamo a calcolare l'entropia di una lingua naturale come l'Inglese. Il repertorio più elementare dei messaggi della lingua inglese è dato dai 26 simboli dell'alfabeto e se tutte le lettere fossero equiprobabili si avrebbe la massima entropia, ovvero

$$H_{en} = \sum_{i=1}^{26} 1/26 \cdot \log_2 \frac{26}{1} = \log_2 26 = 4,7\, bit$$

Tuttavia l'informazione media è molto minore perché le lettere non hanno la stessa frequenza, ovvero la stessa probabilità, e sono vincolate da regole sintattiche alle lettere che le precedono (ad esempio la probabilità di q-u è praticamente 1, la probabilità di j-x è praticamente 0, la probabilità di tio-n è molto elevata). Da una serie di esperimenti[17] si è stimato che l'entropia effettiva della lingua inglese sia inferiore ai due bit per lettera, ovvero meno della metà di quella massima. Ciò significa che in inglese (e con rapporti più o meno simili in tutte le lingue naturali) si usano più del doppio delle lettere strettamente necessarie!
Il rapporto tra l'Entropia effettiva e quella massima è detto *Entropia Relativa* e porta alla formalizzazione del concetto di Ridondanza, secondo la seguente formula:

[17] Per una trattazione sugli esperimenti di stima della entropia delle lingue naturali si veda il lavoro di Fabio G. Guerrero "A New Look at the Classical Entropy of Written English" https://arxiv.org/ftp/arxiv/papers/0911/0911.2284.pdf

$$Ridondanza = 1 - \frac{H_{effettiva}}{H_{massima}}$$

La Ridondanza della lingua Inglese, ed in generale di una lingua naturale è di circa il 50%
La Ridondanza rende la comunicazione più lenta perché richiede più simboli di quelli strettamente necessari, ma permette di correggere errori e omissioni che intervengono nella comunicazione. Essa, secondo la linguista Isabella Chiari[18], nasce nelle lingue naturali da un equilibrio tra una economia della produzione e una della ricezione; le due economie si muovono in direzioni opposte e sono dunque in conflitto:

> "vi è infatti una 'forza' determinata dal desiderio di essere compresi (forza sociale) che conduce all'introduzione della ridondanza, e un'altra 'forza' di pigrizia (forza individuale) che conduce alla brevità e alla semplificazione"

Un equilibrio simile si ritrova nelle comunicazioni digitali: da una parte le tecniche di codifica dei simboli da trasmettere tendono a minimizzare la ridondanza assegnando ai simboli più frequenti codici binari più corti[19], dall'altra, per compensare il rumore del canale di trasmissione[20], si aggiungono dei bit ridondanti che permettono di rilevare eventuali errori.

La ridondanza è anche la caratteristica sfruttata dai crittoanalisti per attaccare un codice ed è quindi l'anello di congiunzione tra la teoria dell'informazione e la teoria della crittografia che Shannon formalizzò nell'anno successivo, ovvero il 1949, pubblicando l'articolo *Communication Theory of Secrecy Systems*[21] dove analizza matematicamente i cifrari noti fino ad allora e formula, tra gli altri, il teorema del **Cifrario Perfetto**: Condizione necessaria e sufficiente per un cifrario perfetto, ovvero immune alla crittoanalisi, è che il numero delle chiavi deve essere uguale al numero dei messaggi. È implicito nel teorema che la chiave sia una **random key** e che sia utilizzata per un solo messaggio: **One Time Pad (OTP)**.

[18] Isabella CHIARI, Ridondanza e linguaggio, un principio costitutivo delle lingue - Carocci editore, 2002

[19] Si veda ad esempio l'algoritmo di Shannon-Fano ulteriormente sviluppato poi da David A. Huffman. Tale algoritmo è ampiamente usato nella compressione dei dei dati (pkzip, jpeg, mp3)

[20] Il secondo teorema di Shannon risolve teoricamente il problema dell'affidabilità della trasmissione su canale rumoroso, definendo le condizioni che permettono di trasmettere informazione nel canale con rumore con probabilità d'errore che può essere resa trascurabile a piacere.

[21] http://netlab.cs.ucla.edu/wiki/files/shannon1949.pdf

Perché tale condizioni si verifichi è necessario che una chiave della lunghezza del messaggio sia preventivamente condivisa dalle due parti in modo sicuro e ciò appare poco pratico. Tuttavia esempi pratici di cifrari perfetti sono stati realizzati: Il sistema di Vernam[22] composto da due nastri di cifre binarie di uguale lunghezza - uno contenente la chiave e l'altro il messaggio - combinati insieme mediante un XOR (Or esclusivo) bit a bit. Il ricevente, in possesso del nastro chiave, può risalire al testo i chiaro eseguendo lo XOR bit a bit tra il testo cifrato e la chiave. A titolo di aneddoto vale la pena ricordare che un cifrario di Vernam fu trovato addosso al Che, dopo la sua uccisione nel 1967. Dal punto di vista matematico, invece, va sottolineato che la tabella di verità dello XOR lo rende una funzione invertibile e di fatto, come vedremo nel prossimo paragrafo, è una delle tre trasformazioni usate anche dai cifrari simmetrici moderni.

A	B	A Xor B
0	0	0
1	0	1
0	1	1
1	1	0

Altri esempi di implementazioni di cifrari perfetti sono stati sviluppati durante la guerra fredda. La **hotline Mosca-Washington**, realizzata nel 1963, era una linea diretta tra il Pentagono ed il Cremlino[23]. Nello stesso periodo le spie Russe (ma anche quelle di altri paesi) usavano un sistema crittografico carta-e-penna, detto cifrario monouso (OTP) che si avvaleva di veri e propri blocchi notes fatti di pagine di caratteri casuali. Ogni pagina rappresentava una chiave; dopo averla usata doveva essere distrutta. Ogni blocco era stampato in due copie: una per la spia (che la doveva conservare nascosta e difenderla a costo della vita) e una per il quartier generale.

8 Crittografia simmetrica moderna

Tra i moderni algoritmi di cifratura simmetrica citiamo l'AES, DES, 3DES, Blowfish. Tra di essi studieremo in dettaglio il DES (Data Encryption Standard) e l'AES (Advance Encryption Standard). Per capire come tutti siano costituiti da tre "mattoni" fondamentali, le funzioni di:

[22]Gilbert Sandford Vernam (1890 – 1960) è stato un ingegnere statunitense e lavorò anch'egli nei Bell Labs

[23]https://www.cryptomuseum.com/crypto/hotline/index.htm

1. Sostituzione
2. Trasposizione (detta anche permutazione)
3. Or Esclusivo (Xor)

8.1 Data Encryption Standard

E' il padre degli algoritmi a chiave simmetrica moderni. Progettato negli anni '70 è diventato Standard nel 1979[24] ed è stato dichiarato insicuro nel 1998 quando fu violato in meno di 60 ore (56 per la precisione) da un computer appositamente costruito[25]. Lo scopo di questo paragrafo è analizzare con degli schemi a blocchi l'algoritmo. Il file viene cifrato a blocchi di 64 bit mediante una chiave di 64 bit (di cui però solo 56 sono effettivi, poiché ogni byte contiene un bit di parità)

La scatola iniziale e quella finale, contrassegnate con T sono delle Trasposizioni dove non interviene la chiave di cifratura. Da essa, il *Keygenerator* produce 16 chiavi distinte, lunghe 48 bit, ognuna delle quali interviene in uno dei 16 round che costituiscono il sottoblocco centrale.
Ognuno dei 16 round è a sua volta costituito dalle seguenti operazioni: Innanzitutto divido i 64 bit di input in due metà. Quella di destra andrà a formare i 32 bit di sinistra dell'outout. La parte di destra, inoltre subisce 5 trasformazioni (invertibili)

1. una Permutazione con Espansione a 48 bit (P-box)
2. una Xor bit a bit con la chiave k_i a 48 bit
3. una Sostituzione con Riduzione a 32 bit (S-box)
4. una Trasposizione

[24]La storia del DES è piuttosto interessante: fu sviluppato, con il nome di **Lucifer** da Horst Feistel, un immigrato di origine tedesca che arrivò in America nel 1934. Durante la seconda guerra mondiale dovette interrompere i suoi studi sulla crittografia per non sollevare sospetti. Dopo la guerra lavorò per l'Air Force's Camridge Research e per la Mitre Corporation ma fu costantemente ostacolato e sabotato dall'NSA finché non arrivò ai laboratori dell'IBM dove negli anni '70 riusci a sviluppare Lucifer. Lucifer si rivelò un sistema robusto ed efficiente, quindi il primo candidato per diventare uno standard commerciale, ma l'NSA fece pressione per limitarne il numero di chiavi possibili a circa 10^{16}, ovvero una lunghezza di 56 bit, rispetto ai 64 bit originali. La versione a 56 bit di Lucifer fu ufficialmente adottata il 23 Novembre 1976 col nome di DES. Vedi [2, Code Book]

[25]Il computer, nominato **Deep Crack**, fu costruito dalla EFF (Electronic Frontier Foudation), con un costo 250.000 dollari. L'anno dopo lo stesso computer violò il DES in 22 ore.

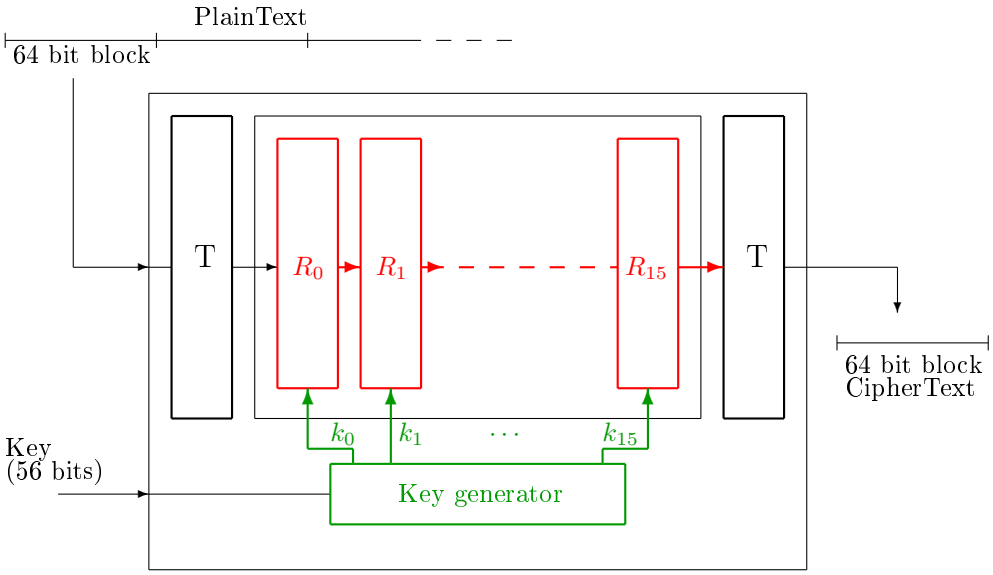


Figura 3: schema a blocchi DES

5. infine una Xor bit a bit con la metà sinistra dell'input

Il procedimento è meglio comprensibile dallo schema 4 dove, viene "aperto" il primo round R_0[26].

8.2 Advance Encryption Standard

Il DES, come detto, è stato dichiarato insicuro dal 1998. E' stato prima sostituito dal triplo DES (3DES), poi dall' AES (Advance Encryption Standard) creato da due crittografi belgi, Joan Daemen e Vincent Rijmen quindi conosciuto anche come Rijndael[27]. Il suo schema a blocchi (vedi figura 5) mostra come la struttura dell'algoritmo sia molto simile a quella del DES; la differenza più evidente è la dimensione dei blocchi, lunghi il doppio, e la lunghezza delle chiavi (128, 192 o 256 bit).

La chiave da 128 va a generare altre 11 sotto-chiavi, una per ogni sottoblocco. Il primo sotto-blocco è un OR esclusivo bit a bit con la prima

[26]Si noti che, per motivi tipografici, rispetto alla figura 3 il disegno è ruotato intorno alla diagonale.

[27]Nel 1997 il NIST (National Institute of Standards and Technology) bandì una gara per trovare il successore del DES come algoritmo standard. Nell'ottobre 2000 è stato scelto come vincitore l'algoritmo Rijndael che prendeva il nome dai due crittologi belgi che lo svilupparono: Joan Daemen e Vincent Rijmen

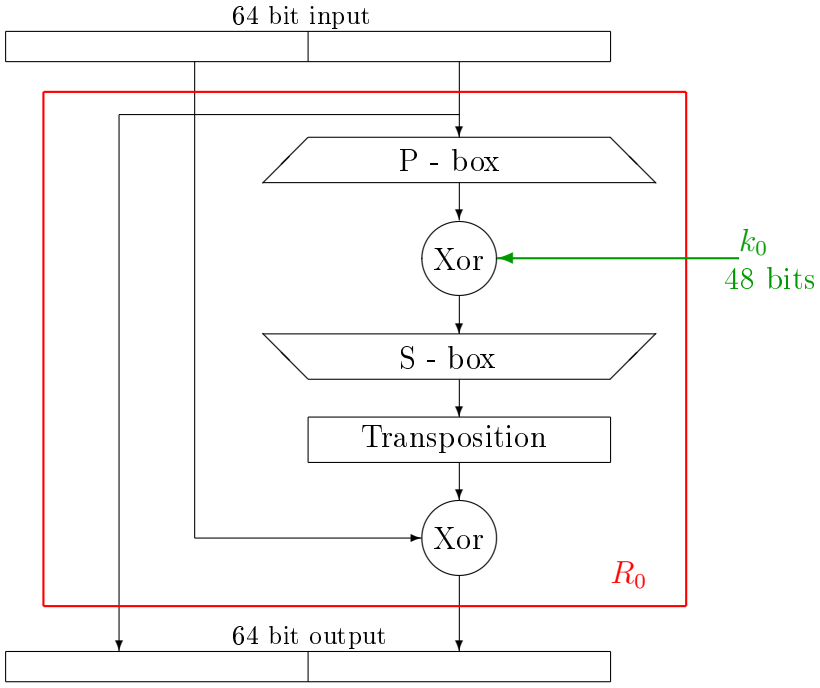


Figura 4: DES: dettaglio del singolo round R_0

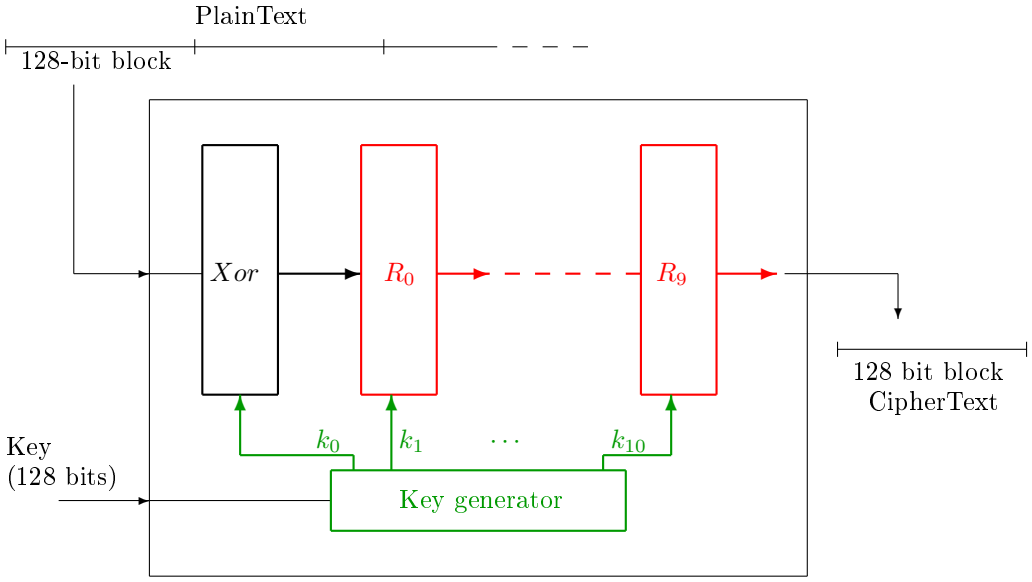


Figura 5: schema a blocchi AES

sotto-chiave, l'uscita del quale entra in una serie di 10 round[28] $(R_0 \cdots R_9)$ ognuno dei quali è costituito dalle solite operazioni elementari[29] come si può notare nella figura 6 che illustra il dettaglio del round $n$esimo.

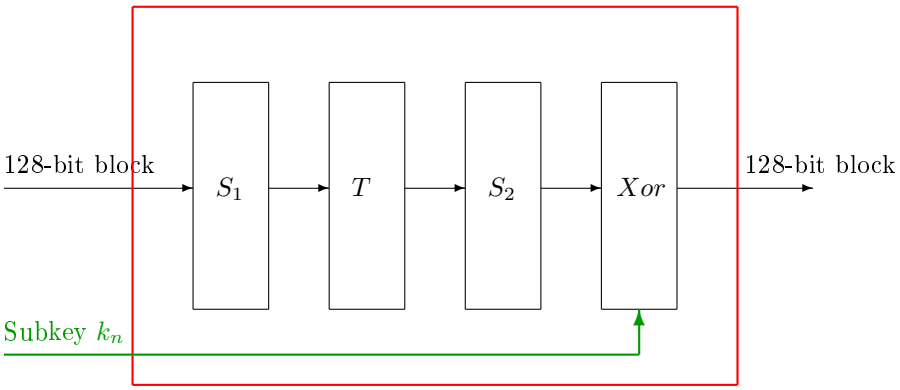


Figura 6: dettaglio round R_n AES

L'input di ogni round, un blocco di 128 bit, subisce 4 trasformazioni:

- Una Sostituzione (S_1) byte a byte. Poiché un byte contiene 8 bit ci sono $2^8 = 256$ combinazioni possibili. Questa parte di algoritmo può essere ottenuta con una tabella $16X16$
- Una Trasposizione (T) realizzata facendo uno shift a blocchi di 4 byte.
- Una Sostituzione (S_2) *doubleword* (4 byte) che prevede quindi $2^{32} = 4G$ combinazioni. E' realizzata con una formula che fa uso dei campi di Galois[30].
- un Ex-OR bit a bit con la sottochiave di 128 bit.

per restituire in output un blocco di 128 bit cifrato, che costituirà l'input del round successivo.

[28] Nelle versioni di AES con chiavi a 192 o 256 bit le iterazioni sono rispettivamente 12 e 14

[29] Le operazioni elementari di crittografia simmetrica sono La Trasposizione, La Sostituzione e L'Ex-OR. La robustezza del sistema si ottiene dalla loro iterazione.

[30] Evariste Galois, matematico francese del 19° secolo, fervente repubblicano, morì a 21 anni in un "duello d'onore" - probabilmente una trappola - dopo aver passato la notte precedente a stendere febbrilmente le note relative alle sue intuizioni matematiche

8.3 Modalità di funzionamento dei cifrari a blocchi

Tutti i cifrari a blocchi come DES, triplo DES ed AES, possono essere usati in diverse modalità, in base a come i blocchi vengono "concatenati" gli uni agli altri. Tali modalità sono state standardizzate dal NIST[31] e qui di seguito se ne dà una breve descrizione.

EBC : Electronic Code Block È la modalità più semplice: ogni blocco viene cifrato in modo indipendente dagli altri. Tale processo è facilmente parallelizzabile ma blocchi uguali di testo in chiaro vengono cifrati negli stessi blocchi di testo cifrato. Questa modalità è indicata per piccoli messaggi, come ad esempio, singoli valori.

CBC Code Block Chain Come suggerisce il nome, ogni blocco di testo in chiaro viene concatenato, mediante or esclusivo (xor) con il blocco appena cifrato. Per cifrare il primo blocco viene utilizzato un vettore di inizializzazione (initialization vector). Questa modalità aggiunge un ulteriore livello di complessità.

CFB Cipher FeedBack È stato introdotto per convertire idealmente un cifrario a blocchi in un cifrario a flusso (per operare in tempo reale).

OFB Output FeedBack Ha scopi simili a CFB.

CRT Counter Viene usato un contatore delle dimensioni dei blocchi. Questa modalità è stata introdotta successivamente alle 4 precedenti per essere applicata ad ATM e IPsec. È utile per requisiti di alta velocità. Fu introdotto da W.Diffie e M.Hellman, due crittanalisti che incontreremo nei prossimi paragrafi.

Entrare nei dettagli delle differenze tra le diverse modalità di funzionamento dei cifrari a blocchi va oltre gli obiettivi di queste note

8.4 Sulla robustezza di un cifrario

Stabilire la robustezza di un cifrario è un problema complesso. Esistono dei criteri matematici per stabilire se il cifrario è debole ma per stimare veramente la sua robustezza bisogna "metterlo sotto pressione" ovvero sottoporlo all'analisi della comunità crittanalitica rendendone pubblico il codice sorgente.

[31] FIPS 81 e FIPS 198 : https://csrc.nist.gov/csrc/media/publications/fips/198/archive/2002-03-06/documents/fips-198a.pdf

Ciò è in accordo con il principio di Kerckhoffs[32] che nella sua versione più sintetica afferma che **La segretezza deve risiedere nella chiave, non nell'algoritmo!**. In altre parole si deve assumere che l'avversario venga a conoscenza, primo o poi, di tutti i dettagli del sistema crittografico usato ma la conoscenza pubblica dell'algoritmo non deve risultare in un possibile indebolimento della sua robustezza crittografica che si basa invece sulla segretezza della chiave.

[32] Auguste Kerckhoffs (1835–1903) fu un linguista, crittografo e matematico olandese autore del manuale "La cryptographie militaire". Non va confuso con Gustav Robert Kirchhoff (1824 – 1887), fisico e matematico tedesco che formulò i principi relativi alle correnti e le tensioni nei circuiti elettrici.

9 Crittografia pubblica

Gli algoritmi simmetrici come l'AES sono molto performanti e robusti, ma presentano il problema della condivisione della chiave segreta (the *key distribution problem*). Alice e Bob dovrebbero incontrarsi personalmente per scambiarsela, oppure usare un canale sicuro diverso da Internet. La soluzione a tale problema è stata trovata negli anni '70 e va sotto il nome di *Crittografia Pubblica*. L'idea è che Bob possieda due chiavi distinte ma correlate tra loro, una pubblica (da comunicare al mondo intero) ed una privata (da custodire con la massima segretezza). Nello schema crittografico di figura 1 $k_C = k_{pubb_B}$ è la chiave pubblica di Bob e $k_D = k_{pri_B}$ è quella privata. Solo k_{pri_B} può decifrare ciò che è stato criptato con k_{pubb_B} (e viceversa).

L'idea guida è di trovare delle funzioni che siano facili da eseguire in un verso, ma estremamente difficili da invertire. È facile moltiplicare tra loro due numeri primi molto grandi ma è estremamente difficile fattorizzare un numero di centinaia di cifre (idea usata dall'RSA). E' facile calcolare $c = a^b (\mathrm{mod} n)$ ma è molto difficile trovare il logaritmo discreto, ovvero risalire a b noti c e n (idea sfruttata dal Diffie-Hellman key exchange). E' facile calcolare una successione di punti su una curva ellittica, ma, dato il punto finale, è difficilissimo trovare il numero di iterazioni necessarie ad ottenerlo (idea usata dalla ECC Elliptic Curve Cryptography). I dettagli matematici li vedremo nei prossimi paragrafi, mentre qui è interessante notare che con tale schema Alice può scegliere una chiave segreta e comunicarla a Bob in modo sicuro pur utilizzando un mezzo altamente promiscuo come Internet.

9.1 RSA

L'agoritmo RSA prende il nome dai suoi tre ideatori: Ronald Rivest, Adi Shamir e Leonard Adleman che lo proposero nel 1977.[33] L'idea di base è che la fattorizzazione di numeri grandi (con più di 100 cifre decimali) nell'algebra modulare è estremamente difficile. Ecco come vengono generate le due chiavi:

1. Bob sceglie due numeri primi grandi p e q e calcola $n = p \cdot q$. n è il modulo.

[33] Per la precisione il metodo era già stato scoperto anni prima dalle agenzie investigative britanniche (GCHQ - Government Communications Headquarters) che non lo pubblicarono per ragioni di sicurezza. Esistono dei documenti, pubblicati solo nel 1997 che mostrano che James Ellis aveva scoperto la crittografia a chiave pubblica e che nel 1973 Clifford Coks aveva scritto un documento interno in cui era descritta una versione particolare dell'RSA. I nostri tre autori, comunque, idearono il loro algoritmo in maniera indipendente.

2. Bob calcola il prodotto $z = (p-1) \cdot (q-1)$ e sceglie e primo rispetto a z ovvero tale che $MCD(z, e) = 1$

3. Bob calcola[34] d, il moltiplicativo inverso modulo z per cui $(d \cdot e \equiv 1 (\mathrm{mod} z))$

4. Bob rende noti e ed n (la chiave pubblica), mentre tiene segreti d, p e q (e di conseguenza z)

5. Alice cifra il messaggio in chiaro come $c = a^e \bmod n$

6. Bob decifra il messaggia con la formula $a = c^d \bmod n$

Vediamo qualche esempio numerico per chiarire le idee. Lavoreremo con numeri estremamente piccoli rispetto alle reali chiavi RSA la cui lunghezza va da un minimo di 1024 bit ad un massimo di 4096 bit

esempio 1 *Proviamo prima con numeri molto piccoli: siano* $p = 97$ *e* $q = 19$. *essi producono* $n = 97 \cdot 19 = 1843$ *e* $z = 96 \cdot 18 = 1728$
scegliamo $e = 23$ *(infatti* $MCD(1728, 23) = 1$*) e usiamo l'algoritmo di Euclide esteso che in 4 iterazioni converge al moltiplicativo inverso di* $23 \bmod 1728$ *ovvero* 1127
Ora Alice cifra il proprio messaggio (ad esempio 105*) con la chiave pubblica* $(23, 1843)$*:*

$$105^{23} \bmod 1843 = 497$$

Quando Bob riceve il messaggio 497 lo decifra con la formula

$$497^{1127} \bmod 1834 = 105$$

si noti che il numero 497^{1127} è un numero elevatissimo, praticamente infinito, rispetto alle chiavi e manderebbe il sistema in overflow. Per calcolarlo è opportuno notare che

$$(a \cdot b)(\mathrm{mod} n) = (a \bmod n) \cdot (b \bmod n)$$

Quindi calcolando la potenza come iterazioni di prodotti si può mantenere limitato a n ogni risultato parziale. Ecco un frammento di codice che permette di calcolare $(a^b)(\mathrm{mod} n)$

```
for (i=0, p=1; i<b; i++) p = (p*a) % c;
```

[34] con l'algoritmo euclideo esteso

esempio 2 *esempio numerico con 4 cifre prendiamo* $p = 3163$ *e* $q = 4999$. *il loro prodotto* $n = p \times q = 15811837$
il prodotto dei loro precedenti $z = (p-1) \times (q-1) = 3162 \times 4998 = 15803676$ *Ora si deve scegliere e primo rispetto a z. Scegliamo un numero primo, ad esempio* 3011 *e calcoliamo il moltiplicativo inverso di e modulo z, ovvero d t.c.* $d \times e = 1(modz)$ *L'algoritmo di Euclide esteso converge in 10 passi e fornisce* $d = 9500051$ *infatti* $e \times d = 28604653561$ *e il resto della divisione con z dà* 1

esempio 3 *esempio numerico con 9 cifre: si prendano* $p = 885320963$ *e* $q = 238855417$

9.1.1 efficienza computazionale

L'algoritmo di cifratura RSA, come abbiamo appena visto, si basa su una formula matematica semplice ($c = a^e \bmod n$) dove però i numeri coinvolti sono enormi, ovvero composti da migliaia di bit (da 1024 a 4096). L'elevamento a potenza a^e, svolto nel modo più immediato, ovvero seguendo la definizione ($a \cdot a \cdots a$, e volte) prevede di eseguire un numero astronomico di moltiplicazioni che richiederebbero un tempo smisurato[35].

Per calcolare a^e si sfrutta perciò la rappresentazione binaria dell'esponente e le proprietà delle potenze. Consideriamo e nella sua rappresentazione binaria:

$$e = b_0 \cdot 2^0 + b_1 \cdot 2^1 + b_2 \cdot 2^2 + b_3 \cdot 2^3 + \cdots + b_t \cdot 2^t$$

Dove $b_i \in \{0, 1\}$ sono i bit che compongono e. Allora

$$\begin{aligned} a^e &= a^{(b_0 \cdot 2^0 + b_1 \cdot 2^1 + b_2 \cdot 2^2 + b_3 \cdot 2^3 + \cdots + b_t \cdot 2^t)} \\ a^e &= a^{b_0} \cdot (a^2)^{b_1} \cdot (a^4)^{b_2} \cdot (a^8)^{b_3} \cdots (a^{2^t})^{b_t} \\ a^e &= a^{b_0} \cdot (a \cdot a)^{b_1} \cdot (a^2 \cdot a^2)^{b_2} \cdot (a^4 \cdot a^4)^{b_3} \cdots (a^{2^{t/2}} \cdot a^{2^{t/2}})^{b_t} \end{aligned}$$

Ricordando che $(a \cdot b)(\bmod n) = (a \bmod n) \cdot (b \bmod n)$ tale formula si presta ad essere implementata con il seguente algoritmo detto *Left-to-Right exponentiation*

```
x ← a
c ← 1
for (i ← 0; i < b; i++)
```

[35] Molto più dell'età dell'universo... Ipotizzando che una normale CPU possa fare 3G moltiplicazioni al secondo, essa impiegherebbe centinaia d'anni ad eseguire l'elevamento a potenza con un esponente a 64 bit; se si considera che il tempo raddoppia ad ogni bit aggiunto, si comprende come il tempo di computazione esploda velocemente verso l'infinito

```
    if(b_i == 1) then
        c ← (c · x) mod n
    endif
    x ← (x · x) mod n
endfor
```

Le operazioni svolte dalle nostre CPU per calcolare $c = a^e \bmod n$ sono quindi proporzionali alla lunghezza t della chiave.

9.2 Diffie-Hellman key exchange

Un'altra soluzione per "scambiare" una chiave segreta attraverso un canale insicuro è rappresentata dal metodo che prende il nome dai due matematici che l'hanno messo a punto: Whitfield Diffie e Martin Hellman [36]. Il protocollo di scambio della chiave Diffie-Hellman (*Diffie-Hellman key exchange*) si basa sulla difficoltà di calcolare il logaritmo discreto modulo n di numeri grandi. Il logaritmo discreto è l'operazione inversa dell'elevamento a potenza in modulo. Ovvero sia n un numero primo e g una sua radice, primitiva[37] dato $b < n$, si dice che h è il logaritmo discreto di b se

$$b = g^h (\mathrm{mod} n)$$

Vediamo come funziona:

1. Alice sceglie p primo (tale che $(p-1)/2$ sia ancora primo) e g una sua radice primitiva (detta anche *generatore moltiplicativo*) p e g possono essere resi pubblici.

2. Alice sceglie a caso un x segreto con $1 < x <= p-2$ e spedisce a Bob g^x

3. Bob sceglie a caso un y segreto con $1 < y <= p-2$ e spedisce ad alice g^y

4. Alice calcola la chiave segreta $K = (g^y)^x$ mentre Bob calcola $K = (g^x)^y$

[36] Diffie ed Hellman furono anche gli autori di un articolo intitolato "Exhaustive cryptanalysis of the NBS data Encrcyption Standard del 1977 in cui ipotizzavano una macchina in grado di violare il DES in 24 ore.

[37] g si dice radice primitiva di un numero primo p se $g^k(modp)$ produce tutte le classi di congruenza non nulle, ovvero, al variare di k da come risultato tutti i numeri compresi tra 0 e $p-1$.

9.3 Curve Ellittiche

Un altro esempio di funzione facile da fare in un verso e difficilissima da invertire è dato dalle successioni di punti sulle curve ellittiche (elliptic curves) ottenute dalla cosiddetta *point addition*. Una generica curva ellittica è rappresentata dall'equazione:

$$y^2 = x^3 + ax + b$$

Dati due punti P e Q sulla curva si definisce la somma (*point addition*) $P+Q$ tracciando la retta passante per P e per Q, intersecando la curva nel punto R e conducendo la verticale per tale punto.

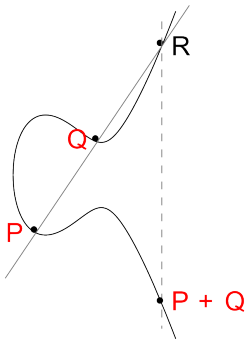


Figura 7: Point Addition

Data tale somma, si capisce che per sommare un punto g con se stesso si deve intersecare la tangente alla curva in g, individuando così $-2g$ da cui, tracciando la verticale si trova $2g = g + g = g \cdot 2$ ottenendo così l'operazione di *point duplication*.

A questo punto possiamo definire la successione di punti $g_0, g_1, g_2, ..g_n$ dove ogni punto $g_k = k \cdot g_0$ è ottenuto sommando g_0 al punto g_{k-1}. Scegliendo opportunamente il punto iniziale g_0 detto generatore[38], i vari punti della successione si distribuiscono uniformemente sulla curva così che risulta molto facile calcolare $g_k = k \cdot g_0$, ma, noto il punto g_k è computabilmente intrattabile risalire al numero k che l'ha generato.

Ci troviamo quinti di fronte al *problema del logaritmo discreto* che infatti è usato per scambiare una chiave attraverso un canale insicuro ricalcando il

[38]Il punto g_0, per essere un buon generatore, non può essere un punto qualsiasi della curva ma deve soddisfare alcune proprietà, ovvero deve generare una successione di punti che non diverge prima di n iterazioni, dove n è un numero molto grande, modulo entro cui si lavora. Si dice allora che g_0 è il generatore di un gruppo ciclico G di ordine n

Diffie-Hellman Key exchange che diventa così *Elliptic Curve Diffie Hellman Protocol* (ECDHP):

1. Alice e Bob concordano i parametri iniziali che sono resi pubblici: i coefficienti della curva a e b ed il punto generatore g_0

2. Alice sceglie la sua chiave segreta d, ovvero un numero intero per il quale moltiplicare g_0. Calcola $P = d \cdot g_0$ e lo manda a Bob attraverso il canale insicuro.

3. Bob sceglie la propria chiave privata e, calcola $Q = e \cdot g_0$ e lo invia ad Alice.

4. Alice e Bob calcolano la chiave condivisa $R = R_A = d \cdot Q = d \cdot e \cdot g_0 = e \cdot P = R_B$

Eve non riesce a calcolare R pur venendo in possesso dei parametri iniziali e dei prodotti P e Q perché non riesce a risalire a d oppure ad e.

Un altro esempio importante dell'utilizzo della ECC è la firma digitale ECDSA (Elliptic Curve Digital Signature Algorithm) la cui spiegazione viene riportata in appendice.

Rispetto all'RSA la ECC offre il notevole vantaggio di avere, a parità di robustezza, chiavi decisamente più corte e quindi complessità computazionale minore e tale che, con l'aumentare della sicurezza richiesta, il rapporto tra le due cresce esponenzialmente. Vedi tabella[39] seguente:

Security bits level	RSA	ECC
80	1024	160
112	2048	224
128	3073	256
192	7680	384
256	15360	512

L'idea di usare le curve ellittiche in crittografia (ECC Elliptic Curve Cryptography) è stata inizialmente proposta da Neal Koblitz e Victor S. Miller nel 1985.

[39]ricavata dalle pubblicazioni "Security Analysis of Elliptic Curve Cryptographyand RSA" http://www.iaeng.org/publication/WCE2016/WCE2016 pagine 419-422.pdf

9.4 Complessità computazionale

La crittografia a chiave pubblica è troppo onerosa in termini computazionali per spedire grosse quantità di dati: basti pensare al numero di operazioni necessarie a produrre un blocco cifrato nel caso dell'algoritmo RSA. La crittografia simmetrica è invece molto più veloce (infatti un blocco cifrato si ottiene con pochi cicli di operazioni elementari) ma presenta il problema di condividere le chiavi da usare. Combinando le due (usando, ad esempio, RSA per scambiare le chiavi segrete e AES per cifrare il traffico) si possono ottenere sistemi in grado di sfruttare i vantaggi e ridurre gli svantaggi di ognuna. Nei prossimi paragrafi analizzeremo alcuni di questi sistemi, detti *ibridi*, dopo aver trattato la terza famiglia di funzioni crittografiche fondamentali: le funzioni di HASH[40]

[40]Come vedremo è improprio definire l'HASH una funzione crittografica perché non serve a nascondere un messaggio, ma a calcolarne una sintesi. Essa serve principalmente per verificare l'integrità di un messaggio.

10 Message Digest

Il Message Digest (sintesi di un messaggio) è l'output di una funzione di *HASH* che *rimescola e trita*[41] i bit di un messaggio datole in input.

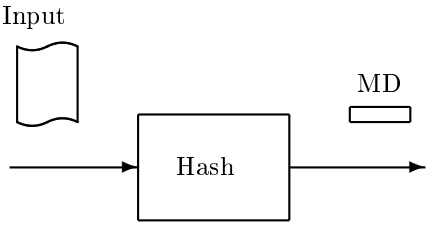


Figura 8: funzione di HASH

Essa è una funzione unidirezionale o entropica (strictly one way function) che deve soddisfare le seguenti proprietà:

1. (deterministic): dato lo stesso input si otterrà sempre lo stesso output
2. (efficient): è computazionalmente veloce;
3. (fixed-length output) la lunghezza dell'output è fissa (tipicamente 128, 160, 256 o 512 bit) indipendentemente dall'input;
4. (avalanche effect) cambiando anche un solo bit nel messaggio di input, l'output cambia completamente;
5. (pre-image resistant): è computazionalmente intrattabile trovare un messaggio il cui MD corrisponda ad uno dato;
6. (collision resistance): è computazionalmente intrattabile trovare due messaggi diversi che diano lo stesso MD. Nella versione forte[42] di questa proprietà, non esistono m1, m2 tali che hash(m1) = hash(m2).

Va notato che rispetto alle funzioni crittografiche fin qui viste, la funzione di hash ha due importanti differenze peculiari:

- non ha una chiave di cifratura

[41] dall'inglese to hash che significa "sminuzzare" "pasticciare"

[42] La versione forte della proprietà (collision-free) non è ottenibile in quanto la lunghezza dell'output è fissa. Sia n il numero di bit che costituiscono l'output, si possono ottenere 2^n diverse combinazioni quindi per $2^n + 1$ documenti ci sarà sicuramente almeno una collisione. Tale fatto è utilizzato dall'attacco del *compleanno* che sfrutta l'omonimo paradosso probabilistico.

- non è invertibile

La funzione di HASH può quindi essere vista come una sofisticata *checksum* che permette di verificare l'integrità del file ricevuto confrontando il message digest ricevuto con quello calcolato. Viene implementata scomponendo il file di input in blocchi di lunghezza fissa e applicando ad ogni blocco funzioni matematiche di base bit a bit quali SUM, AND, OR, XOR, ROT, per alcune decine di round. Si può pensare agli schemi visti per gli algoritmi di crittografia simmetrica dove però, anziché concatenare i blocchi in uscita, essi vengono combinati per ottenere un output di lunghezza fissa. Dal punto di vista della sicurezza informatica, la funzione di hash viene tipicamente usata per salvare il MD delle password, anziché la password in chiaro. Si veda ad esempio il file `/ect/shadow`[43]

10.1 attacchi al message digest

La funzione di Hash è per definizione non invertibile: l'attacco più banale (e computabilmente più costoso) ad un file che contiene l'hash delle password è il *brute force* : ovvero generare tutte le possibili stringhe, calcolarne per ognuna il message digest e confrontarlo con quelli del file. Un attacco più efficace è quello del *dizionario*: anziché generare tutte le possibili stringhe di input, le si prende da un elenco che contiene le parole di uso comune e le password più frequenti[44]. Attacchi molto più efficienti sono quelli che usano tabelle dove sono state raccolte le password più comuni e ne sono già stati calcolati i message digest. Queste tabelle si chiamano *hash tables* o *rainbow tables*[45].
Per evitare attacchi di tipo *lookup tables* si aggiunge alla password un po' di *sale* ovvero una stringa psuedo-casuale da aggiungere alla password in chiaro prima di calcolarne il message digest. A tale scopo si usa un Cryptographically Secure Pseudo-Random Number Generator (CSPRNG). si noti che poiché il sale deve essere memorizzato insieme al message digest, salare una password previene il lookup table attack ma non l'attacco del dizionario o brute force.
L' *attacco del compleanno* o *collision attack* è invece un attacco al controllo di integrità di un file. Si basa sul *paradosso del compleanno* per cui la probabilità che, in un gruppo di persone, due compiano gli anni lo stesso giorno

[43] per sopportare un attacco indicizzato viene spesso aggiunto un pizzico di sale in input...

[44] Questo è il motivo per cui una password robusta non deve essere una parola di uso comune, deve avere una lunghezza minima e comprendere cifre e caratteri speciali.

[45] Le rainbow tables sono delle hash table compresse per risparmiare spazio di archiviazione: ne risulta una velocità computazionale ridotta.

è molto più alta di quanto il senso comune faccia pensare. Nel caso di 30 persone la probabilità[46] è superiore al 70%. Questo fatto viene sfruttato per produrre due file (uno originale ed uno fraudolento[47]) che abbiano lo stesso message digest. Nei due file individuano un numero sufficiente di punti (ad esempio 30) modificabili senza che il documento cambi in modo percettibile: ad esempio si può agire su virgole, spazi o righe vuote. Così si possono generare moltissime varianti dei due documenti (nel caso specifico 2^{30}) per le quali la probabilità che si verifichi almeno una *collisione* (ovvero di avere lo stesso message digest) è tanto maggiore quanto più corta è la lunghezza del message digest stesso. Senza inoltrarci ulteriormente nella matematica statistica, è utile ricordare un attacco, il *Flame malware collision attack*, dove il malware era stato firmato in modo fraudolento sfruttando la bassa resistenza alle collisioni dell'algoritmo MD5. Questo particolare attacco si potrà analizzare meglio dopo aver trattato la firma digitale, i certificati X.509 e le PKI.

11 La firma digitale

La firma digitale si ottiene cifrando con la chiave privata la sintesi (o message digest) del documento. Si ottiene così l' *autenticità* e l' *integrità* del documento (non la segretezza). Quando si riceve il documento (che è in chiaro) si decifra la firma con la chiave pubblica del mittente (Alice), ottenendo così il message digest da confrontare con quello calcolato. Se sono identici il documento è autentico (perché solo alice può aver cifrato il message digest) e integro (perché una modifica anche piccola avrebbe dato luogo ad un message digest calcolato diverso da quello ricevuto).

[46] In particolare la distribuzione statistica del problema del compleanno, se si assume che tutti i giorni dell'anno siano equiprobabili per una nascita, ha un profilo crescente che raggiunge il 50% per un gruppo di 23 persone e supera il 90% con 45 persone.

[47] Un caso notevole è quello di due contratti con due importi diversi. Quello originale potrebbe indicare una somma esigua, mentre quello fraudolento una somma molto elevata.

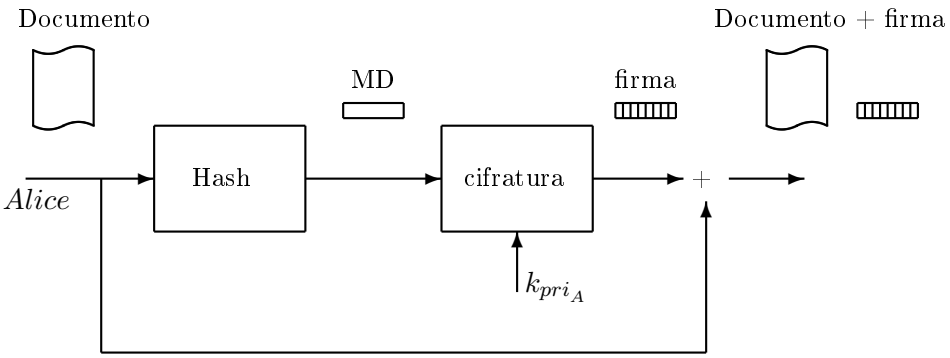


Figura 9: schema della firma digitale

12 HMAC

Oltre alla firma digitale vista nel paragrafo 11 esiste un altro sistema per garantire autenticità ed integrità ai documenti: L'HMAC. **keyed-Hash Message Authentication Code** è una tecnica che utilizza una funzione di Hash che *mescola* insieme al file di input una chiave *segreta* per la generazione del message digest che quindi dipende sia dal testo originale che dalla chiave segreta.

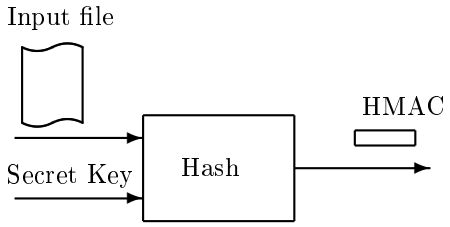


Figura 10: funzione di HMAC

La principale Differenza sta nel fatto che nell'HMAC la chiave segreta è simmetrica e quindi deve essere preventivamente concordata *preshared key*, mentre la firma digitale di basa sulla coppia di chiavi pubblica/privata e su una PKI (Public Key Infrastructure).

Un confronto tra i due modi di "firmare" i dati può essere sintetizzato come segue:

a) HMAC è molto più veloce rispetto alla firma con algoritmo asimmetrico

b) HMAC necessita di un metodo per scambiare la chiave segreta (Diffie-Hellman o RSA)

c) HAMC non identifica un solo soggetto, ma una connessione. E' infatti usato in IPsec[48]

[48]IP security è una re-implementazione del protocollo IP che aggiunge autenticità, riservatezza e integrità al livello Network. Vedi RFC 6071 : IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap

13 Sistemi Crittografici Complessi o Ibridi

Per ottenere dei sistemi crittografici "completi" si integrano le funzioni crittografiche fondamentali viste nei paragrafi precedenti. La firma digitale è già un primo esempio in cui crittografia pubblica e funzioni HASH sono utilizzate per ottenere integrità e autenticità. Studieremo nei prossimi paragrafi il PGP, l'https, TOR e la moneta elettronica Bitcoin.

13.1 Pretty Good Privacy

Per lo scambio Riservato, Integro ed Autentico delle e-mail è stato ideato, da Phil Zimmermann, nel 1991, il sistema PGP (Pretty Good Privacy). Se ne riporta qui di seguito lo schema a blocchi, particolarmente interessante come sintesi delle tre tecniche crittografiche viste: quella a chiave pubblica, quella a chiave privata e il message digest

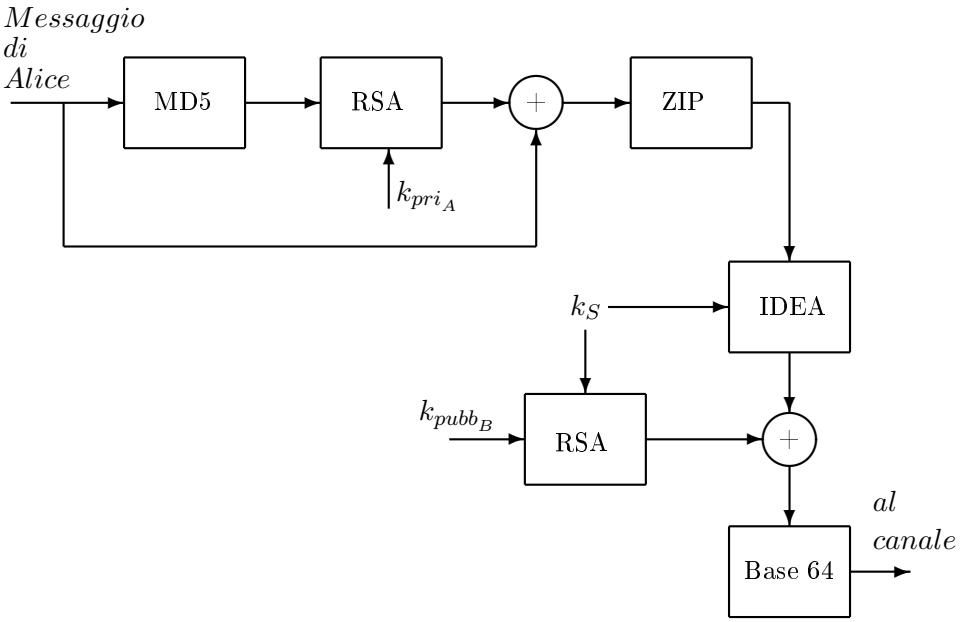


Figura 11: schema PGP

In ricezione, chiaramente, Bob dovrà applicare uno schema inverso per estrarre la chiave di sessione k_S mediante algoritmo RSA e la propria chiave privata (k_{pri_B}) e potrà perciò decifrare (e scompattare) il messaggio con firma digitale allegata. Per verificarne autenticità e integrità decifrerà il MD

mediante RSA con la chiave pubblica di Alice (k_{pubb_A}) e lo confronterà con quello da lui calcolato.

13.1.1 GPG ed Enigmail

Il sistema PGP è un sistema coperto da copyright proprietario[49] ma vi sono diverse versioni open source che si attengono allo standard OpenPGP. Tra queste la più famosa è la GPG (Gnu Privacy Guard) rilasciata con licenza GNU GPL della Free Software Foundation. Il plug-in **Enigmail** offre una facile implementazione di gpg al client di posta Thunderbird.

[49] L'iter economico/legale della licenza software di PGP è piuttosto complesso. E' curioso ricordare che agli inizi Zimmermann fu addirittura incriminato dal governo degli Stati Uniti d'America per esportazione di armi da guerra senza apposita licenza.

13.2 HTTPs

Per conferire riservatezza, integrità e autenticità al protocollo HTTP si è aggiunto uno strato crittografico : il SSL/TLS (Secure Socket Layer / Transport Layer Security) che lavora sopra il TCP per concordare la suite crittografica, verificare l'identità del server, quindi scambiare una chiave di sessione (handshake) con la quale cifrare il traffico.
Dopo che il TCP ha aperto la connessione con il server, si verifica quindi il TSL handshake secondo il seguente diagramma di sequenza:

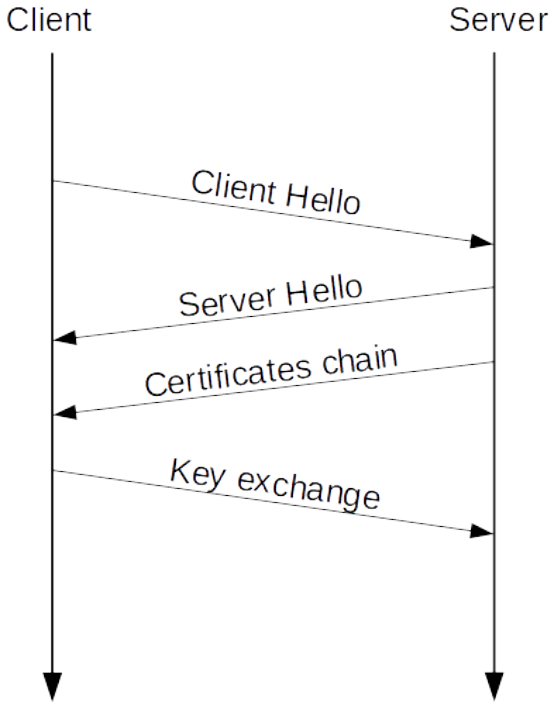


Figura 12: ssl/tls handshake

Client Hello Il client richiede l'apertura di una connessione sicura inviando la suite crittografica di cui dispone[50].

Server Hello Il Server specifica quale suite verrà usata (tra quelle disponibili al client).

[50] oltre ad altri parametri come la versione di protocollo, un ID di sessione e i metodi di compressione

Certificate Chain Il Server invia il proprio certificato (o la catena di certificati) contenente la chiave pubblica.

Key Exchange Grazie al certificato il client può verificare l'autenticità del server (vedi prossimo paragrafo) e grazie alla chiave pubblica può condividere una chiave di sessione generata in modo pseudocasuale. Quindi opera il **Change CipherSpec** ovvero il passaggio al cifrario simmetrico per poter finalmente iniziare lo scambio dati.

Nota: lo schema appena mostrato indica solo lo scambio di pacchetti essenziale e trascura ad esempio pacchetti di tipo **Server Hello Done** che ha lo scopo di indicare la fine di tale fase. Lo schema inoltre Potrebbe essere ancor più complesso se anche il browser dovesse attestare la propria identità. In tal caso anch'esso invierebbe il proprio certificato.

13.3 Certificati X.509 e PKI

Come può il browser essere certo che la chiave pubblica ricevuta sia proprio quella del server che intende contattare e che non sia stata invece la solita *Eve* ad interporsi nella comunicazione, inviando una sua chiave pubblica?

Per verificare l'identità del server, il browser deve interpellare una terza entità, una Certificate Authority (CA) di cui si fida (o si deve fidare) che garantisce l'associazione tra quel server (o più in generale tra un "soggetto") e la sua chiave pubblica. Per fare ciò la CA firma il Certificato con la propria chiave privata e il client può verificarne la firma perché contiene già le chiavi pubbliche di tali CA all'interno di certificati autofirmati (vedi ad esmpio su Firefox –> Menù –> Opzioni –> Privacy e sicurezza –> Mostra certificati). Un certificato X.509 è un documento standardizzato che contiene, oltre al nome del soggetto e la sua chiave pubblica, le date di validità, il nome dell'autorità certificante, l'identificativo dell'algoritmo di firma e la firma stessa. Nella tabella 2 si possono vedere i campi più significativi del certificato di wikipedia.

Si Noti che la firma del certificato è ottenuta con algoritmo RSA e che la chiave pubblica è relativa alla crittografia Ellittica.
Un certificato potrebbe essere firmato da una CA intermedia la quale a sua volta possiede un certificato firmato da una CA root. In questo caso il server invia una catena di certificati in modo che il client possa verificarne l'autenticità. L'intero sistema prende il nome di **PKI : Public Key Infrastructure**.

Nome campo	valore
Nome soggetto	*.wikipedia.org
Nome autorità Emittente	DigiCert inc.
Validità	dal 12/11/19 al 6/10/2020
Chiave pubblica	256 bit (con Algoritmo Elliptic Curve
algoritmo di firma:	SHA-256 with RSA Encryption
...	...

Tabella 2: Alcuni campi del certificato X.509 di wikipedia

13.3.1 web of trust

Lo stesso problema si pone anche in PGP: Alice e Bob devono conoscere le rispettive chiavi pubbliche, ma come essere certi della loro autenticità? Anche in questo caso ci si affida a dei certificati ma anziché essere firmati da una autorità riconosciuta, essi vengono validati in modo paritetico dagli utenti della rete secondo uno schema noto come **rete di fiducia**. In tale sistema i certificati sono firmati dagli utenti stessi della rete che attestano così di conoscere il proprietario di un certificato e di avere fiducia in lui[51]. Un certificato firmato da diverse persone di cui mi fido diventa così credibile, un po' come nella vita reale ci fidiamo di un idraulico perché diverse persone di cui abbiamo fiducia ce l'hanno raccomandato. Si può dire che il web of trust sia contrapposto alla PKI in quanto li primo è puramente paritetico e la seconda completamente centralizzata. Recentemente si sta imponendo un modello che sta a metà strada: il **SPKI: Simple Public Key Infrastructure** che ha degli enti certificanti a livello locale.

[51]Si può firmare un certificato con diversi livelli di fiducia: elevato, medio, basso, generico. quando si firma con livello elevato significa che si dichiara di aver controllato scrupolosamente l'identità del possessore, ad esempio incontrandolo personalmente. Io, ad esempio ho firmato i certificato del prof. Bedani.

13.4 The Onion Route

PGP e https sono ottimi sistemi per garantire confidenzialità, integrità e autenticità nelle comunicazioni, ma non garantiscono l'anonimato. Di fatto i meta dati[52] della comunicazione non sono (e non possono) essere cifrati quindi sono a disposizione dei sistemi di sorveglianza estesa[53].

Per ottenere l'anonimato in Internet si può usare TOR (The Onion Router): una rete di nodi che fungono da intermediari tra mittente e destinatario. L'idea alla base dell' onion routing[54] è di incapsulare il messaggio in tre o più cassette di sicurezza[55] una dentro l'altra, come scatole cinesi, ognuna con una chiave diversa.

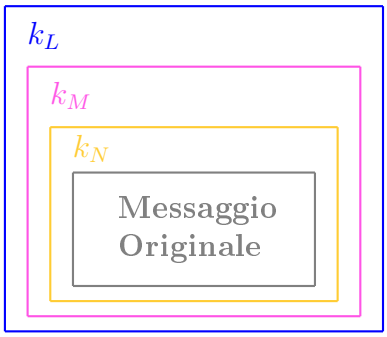


Figura 13: strati di cifratura TOR

Vediamo come funziona.

1. Alice sceglie Lisa, Mary e Noel tra un insieme di nodi intermediari (non necessariamente fidati) detti *TOR nodes* e stabilisce con loro tre

[52] Con il termine di meta dati si indicano quei dati che non fanno strettamente parte del messaggio. Se pensiamo alla posta ordinaria i meta dati sono gli indirizzi del mittente e del destinatario e la data di spedizione. In internet i meta dati sono gli indirizzi IP, le porte TCP/UDP, i timestamp, la geolocalizzazione, ect. La questione della sorveglianza di massa si è imposta da quando il costo della memorizzazione dei dati è crollato e le corporation, così come i governi, hanno cominciato a memorizzare ogni dato prodotto dalla notra attività digitale.

[53] Per una comprensione del fenomeno della *digital surveillance* si consiglia la lettura del saggio di Bruce Schneier[?] "Data and Goliath".

[54] Il termine onion routing fu coniato dai tre ricercatori Goldschlag, Reed, e Syverson e comparve per la prima volta nell'articolo "Hiding Routing Information" del 1996.

[55] La crittografia moderna offre un equivalente digitale alla cassetta di sicurezza: l'involucro crittografico.

rispettive chiavi segrete (k_L, k_M e k_N), scambiate con un metodo di crittografia pubblica, ad esempio il Diffie-Hellman

2. Alice applica tre livelli di cifratura al messaggio originario m come illustrato nello schema di figura 14. Più precisamente il messaggio m e l'indirizzo di Bob (che indicheremo con B[56]), vengono cifrati con la chiave segreta k_N condivisa con Noel. Il tutto viene poi aggiunto all'indirizzo N di Noel e cifrato con la chiave k_M di Mary. L'output di tale cifratura viene accodato all'indirizzo di Mary (M) per diventare l'input della terza cifratura, eseguita con la chiave k_L di Lisa. Il tutto viene finalmente inviato a Lisa.

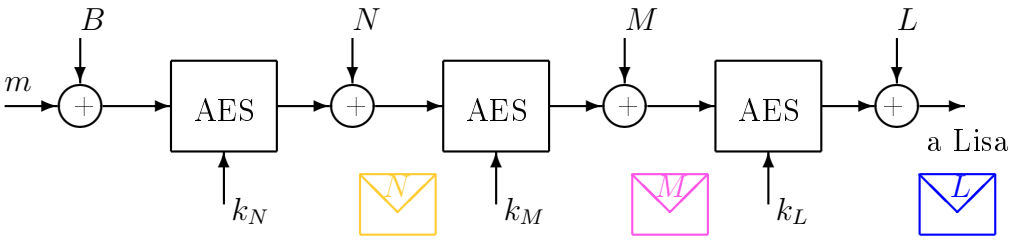


Figura 14: Cifratura TOR

3. Lisa riceve il file cifrato blu, che può essere pensato come una cassetta di sicurezza chiusa dal lucchetto la cui chiave k_L è nelle mani di Lisa perché precedentemente scambiata con Alice. Lisa decifra il file (apre la cassetta) ottenendo l'indirizzo M di Mary e il file cifrato rosa (che non può decifrare), e lo invia ad M

4. Mary esegue operazioni analoghe a quelle svolte da Lisa nel passo precedente, decifrando il file rosa con K_M e ottenendo l'indirizzo N di Noel e il file cifrato giallo che invierà ad N

5. Noel decifra il file giallo con k_N ottenendo l'indirizzo B di Bob e il messaggio m da inviargli.

6. Bob riceve m da Noel ed eventualmente risponde con un messaggio r che spedirà a Noel perché segua il percorso inverso (detto *circuito*), subendo uno strato di cifratura ad ogni passaggio: Noel lo cifrerà con

[56]Gli indirizzi dei cinque soggetti (Alice, Bob, Lisa, Mary e Noel) della comunicazione, in Internet sono indirizzi IPv4 oppure IPv6, ovvero numeri di 32 o 128 bit. In questa trattazione li indicheremo semplicemente con delle lettere: A, B, L, M, N

k_N e lo spedirà a Mary che lo cifrerà con k_M e lo inoltrerà a Lisa che lo cifrerà con k_L che lo recapiterà finalmente ad Alice. Alice potrà rimuove i tre strati di cifratura (perché in possesso delle tre chiavi) e leggere la risposta r

La figura 15 mostra i tre nodi TOR attraversati dal pacchetto cifrato. Il primo nodo (Entry Node), col quale Alice condivide k_L toglie il primo strato di cifratura, il secondo nodo (Relay Node), col quale Alice condivide k_M, toglie il secondo strato di cifratura, quindi il terzo nodo (Exit Node) toglie l'ultimo strato di cifratura mediante k_N. Va notato che ogni nodo può conoscere solo i meta dati relativi al nodo precedente e quello successivo, garantendo di fatto l'anonimato.

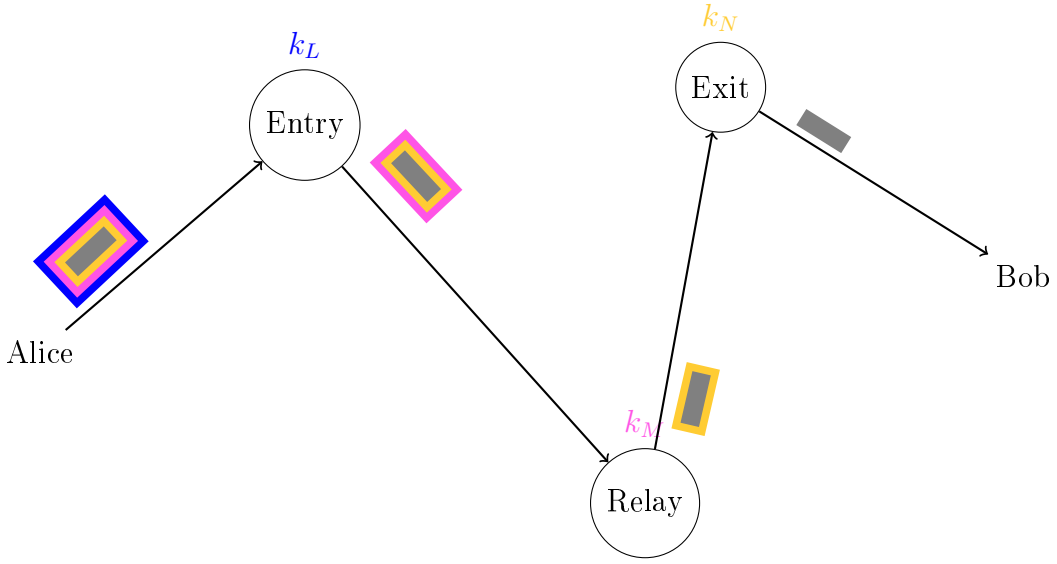


Figura 15: circuito TOR

14 Blockchain e Bitcoin

Un importante e originale esempio di applicazione delle funzioni crittografiche sono le cripto-valute dove la crittografia non viene impiegata per segretare ma per attestare la proprietà di un valore, firmare una transazione e rendere immutabile una struttura dati. Di fatto queste tecnologie usano le funzioni di hash e la crittografia a chiave pubblica. Tra le cripto-valute la più famosa è il **Bitcoin**, nata nel 2008 ad opera del misterioso Satoshi Nakamoto[57] che ne descrisse l'algoritmo nel Libro Bianco *Bitcoin: A Peer-to-Peer Electronic Cash System[8]*[58].

Il cuore del Bitcoin è la **Blockchain**, ovvero la catena di blocchi dove viene registrata immutabilmente ogni transazione. Il termine blockchain è apparso per la prima volta nel codice sorgente (open source) del Bitcoin[59].

Può essere pensata come un *libro mastro*. La blockchain è di fatto un database condiviso (replicato su tutti i nodi della rete Bitcoin), immutabile, composto da blocchi concatenati dove il nodo n-esimo contiene l'HASH del noto (n-1)-esimo.

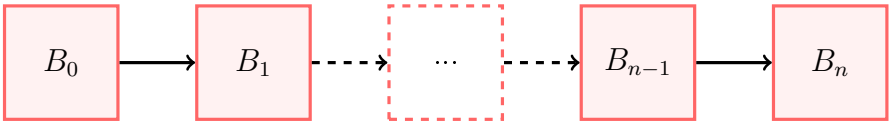


Figura 16: Catena di blocchi

La blockchain può quindi anche essere definita come una rete paritetica che condivide il "libro mastro" delle transazioni (ledger): ogni pagina del libro mastro è rappresentata da un blocco della blockchain. La **transazione** (transaction) è la registrazione dello scambio di valore tra due nodi della rete (può essere denaro, come appunto nel caso delle crypto-valute, ma si può trattare di beni materiali o immateriali, o anche di fasi lavorative...) Ogni blocco è costituito da una intestazione ed un corpo e contiene più transazioni. Vediamo prima come avviene una transazione e poi come viene inserita nel libro mastro.

[57] Satoshi Nakamoto è uno pseudonimo dietro cui si nasconde probabilmente un gruppo di sviluppatori. Nel 2011, comunque scrisse il suo ultimo messaggio dichiarando di aver lasciato il gruppo di sviluppatori per passare al altri progetti.

[58] Il testo, disponibile in diverse lingue, si può trovare al link https://bitcoinorg/en/bitcoin-paper

[59] Il codice sorgente Bitcoin è disponibile su GitHub https://github.com/Bitcoin/Bitcoin/tree/master/src

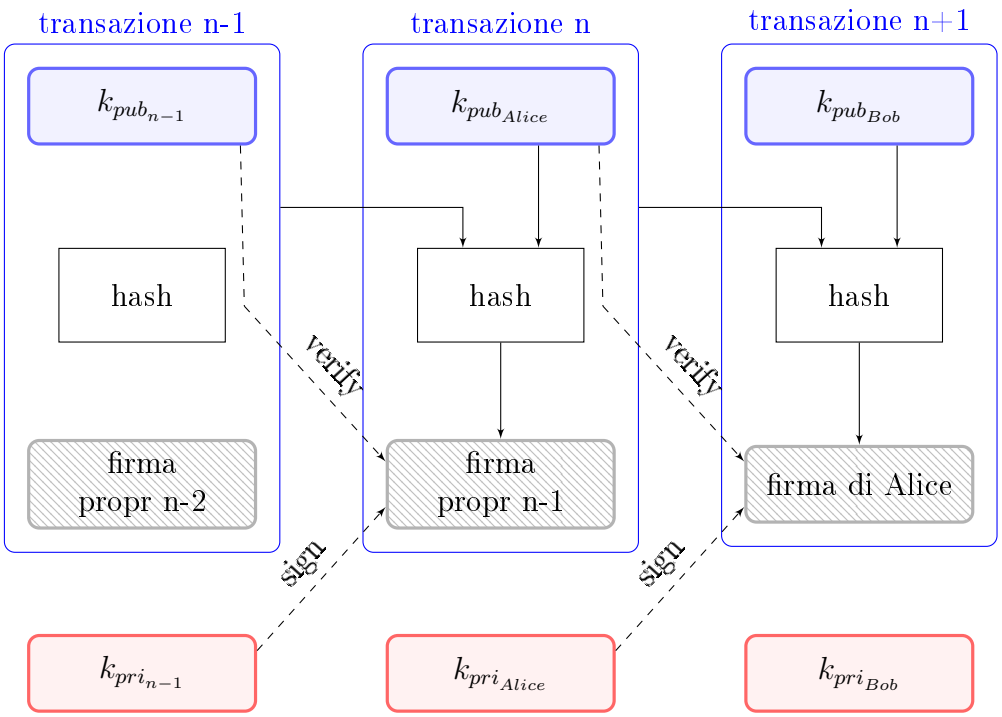


Figura 17: schema di principio di una transazione da Alice a Bob

Una transazione è il passaggio di un valore da un utente ad un altro, ovvero il trasferimento dello *spending control* da un'entità ad un'altra. Consideriamo, come al solito, i nostri eroi, Alice e Bob, entrambi forniti di una coppia di chiavi asimmetriche (pub, pri). Alice vuole trasferire a Bob un valore precedentemente acquisito. Per farlo crea una nuova transazione contenente il valore acquisito e lo firma digitalmente (mediante ECDSA) includendo nella funzione di HASH la chiave pubblica di Bob. Si veda la figura17 che è una rivisitazione dello schema di principio illustrato da Satoshi Nakamoto nel libro bianco, da cui citiamo la spiegazione:

> "Each owner transfers the coin to the next by digitally signing a hash of the previous transaction and the public key of the next owner and adding these to the end of the coin"[8].

quella descritta in figura 17 è la semplificazione di una transazione con un solo input ed un solo output. In generale una transazione può avere più input e più output ognuno dei quali ha dei campi che lo identificano e lo definiscono. Ogni output non ancora speso di una transazione precedente

destinato alla transazione corrente è detto **UTXO (Unspent Transaction Output)**. Nella figura 18 viene dettagliata la struttura di una transazione:

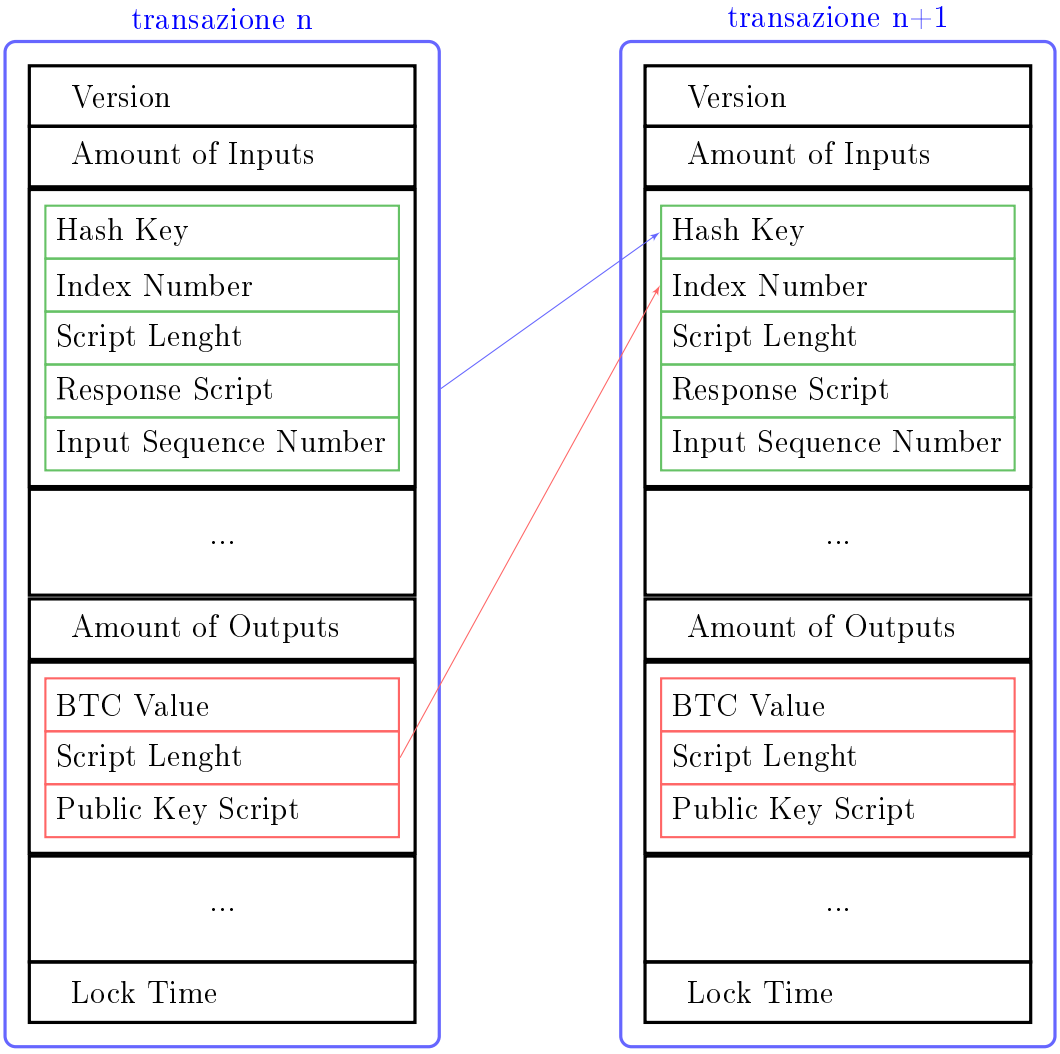


Figura 18: dettaglio dei campi di una transazione Bitcoin

La struttura dati[60] di una transazione prevede alcuni campi globali quali:

Version : versione del protocollo (attualmente 1).

Amount of Inputs numero di inputs, ovvero di UTXOs.

Amount of Outputs numero totale do output.

[60] https://blog.brakmic.com/bitcoin-internals-part-2/

Lock time Un numero che indica il tempo prima del quale la transazione non deve essere minata.

Quindi contiene un campo vettore di strutture input, tante quante specificate nel campo Amount of Inputs. Ogni struttura di input contiene i campi (evidenziati in verde):

Hash Key : il message digest (SHA-256) della transazione precedente

Index Number : intero a 4 byte che indica quale UTXO, tra quelli della transazione precedente è diventato input della attuale transazione

Script Lenght : lunghezza dello script di sblocco (vedi campo seguente)

Response Script : codice[61] che prova che tutte le condizioni associate alla UTXO sono soddisfatte. Il più delle volte si tratta di una firma che attesta la proprietà della UTXO da parte del mittente (anche detto pagante).

Input Sequence Number : campo addizionale probabilmente pensato per conferire maggiore flessibilità al sistema, principalmente per qual che concerne il tempo di mining. Possiamo ignorare il ruolo di questo campo in prima approssimazione.

In base del campo Amounts of Outputs vi è poi un vettore di strutture di Output, ognuna delle quali composta di tre campi, evidenziati in rosso in figura 18:

BTC value : valore in Bitcoin (BTC) della UTXO. Per la precisione il valore è espresso in Satoshi, il minimo valore trasferibile. 1 BTC equivale a 100 milioni (10^8) di Satoshi[62].

Script Lenght : analogamente a quanto accade per gli input questo campo indica la lunghezza dello script che segue

[61] Il codice con cui è scritto questo script è scritto in bitcoin scripting language: un vero e proprio linguaggio di programmazione, seppur non Turing completo (manca ad esempio dei loop) mediante il quale vengono scritte le condizioni di spendibilità della UTXO associata. Per tale motivo il Bitcoin e le cripto-valute in generale sono dette monete programmabili.

[62] Ad oggi (metà dicembre 2020) un Satoshi equivale a circa 0.02 centesimi di euro quindi un BTC equivale a circa 20 mila euro. Il valore di tale cambio è molto variabile: ad esempio in aprile 2020 si aggirava sui 6 mila euro e nel 2017 era inferiore ai mille euro. Per un grafico sull'andamento del cambio BTC/EUR si veda https://www.money.it/+Bitcoin-Euro-Quotazione+

Public Key Script : detto anche “signature script” o "output script". Nel caso più frequente[63] questo script contiene l’indirizzo bitcoin del ricevente[64]. Più in generale indica le condizioni di blocco che dovranno essere soddisfatte dal corrispondente script di sblocco o *Response Script* in una dialettica di tipo challenge/response.

La concatenazione tra le transazioni, differentemente dalla blockchain che ha struttura seriale, forma una rete che la figura 19 cerca di esemplificare:

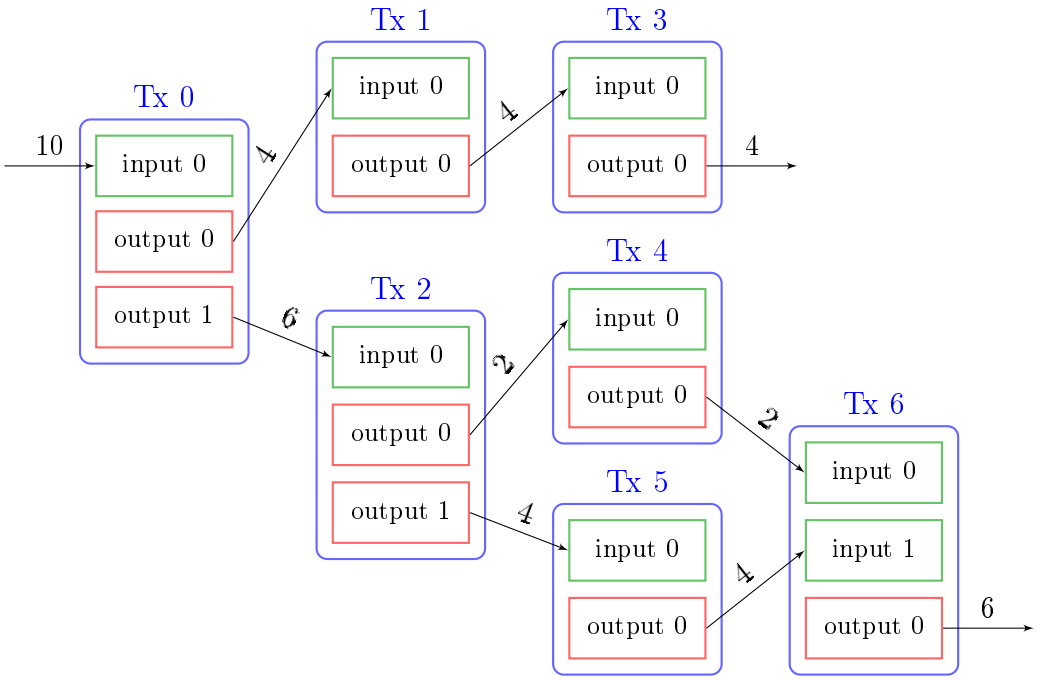


Figura 19: le transazioni sono collegate tra loro come in una rete

Nella figura 19 si nota un perfetto bilanciamento tra input e output. In realtà l’output di una transazione è minore o al più uguale all’input. La

[63] La transazione più comune contiene script di output di tipo Pay-to-Public-Key-Hash (P2PKH).

[64] Un indirizzo bitcoin è generato a partire dalla chiave pubblica dell’intestatario del conto (Bob) attraverso un doppio HASH : sha256 + RIPMED160 codificato poi in Base58Check. "Base58 è un insieme di schemi di codifica da binario a testo, specificati da Satoshi Nakamoto per la rete Bitcoin, al fine di rappresentare numeri interi grandi come testo alfanumerico. Da allora, è stato applicato ad altre critto-valute e applicazioni. È simile al Base64, ma è stato modificato per eliminare sia i caratteri non alfanumerici che quelle lettere che potrebbero essere confuse con altre, quando stampate" [wikipedia]

differenza è la mancia (fee) lasciata al minatore. L'operazione di minare (mining) ed il ruolo dei nodi minatori è descritto nel prossimo paragrafo.

14.1 Aggiungere un blocco alla catena

Una transazione, per essere confermata, deve essere inviata (in broacast) alla rete Bitcoin per essere inserita nella blockchain, ovvero inserita nel libro mastro condiviso. Prima di proseguire, vale la pena riassumere i passi necessari a creare una transazione completa[65]:

1. **Bob**, il ricevente, **crea una chiave privata** da cui deriva (mediante curve ellittiche) la corrisponde chiave pubblica, dalla quale, a sua volta, viene calcolato l'indirizzo bitcoin.

2. **Bob invia ad Alice**, la pagante, **il proprio indirizzo bitcoin**, ovvero il doppio hash della sua chiave pubblica.

3. **Alice crea la transazione** inserendo i dati nella struttura rappresentata in figura 18; in particolare inserisce l'indirizzo bitcoin di Bob nel Public Key Script e specifica negli input gli UTXO necessari al pagamento. La commissione tipica è di 1000 Satoshi.

4. **Alice trasmette la transazione appena creata a tutti i nodi** della rete Bitcoin. Bob, in quanto nodo della rete, vede immediatamente sul suo software la transazione avvenuta, ma prima di considerare il pagamento effettuato attende che la transazione sia inserita nella blockchain.

5. **La transazione viene inserita nella blockchain** da uno dei nodi minatori. Per farlo deve prima risolvere una sfida computazionale (che richiedere in media 10 minuti) e inserire nel blocco la soluzione di tale sfida così che tutta la rete la possa verificare e riconoscere come valida. Per tale lavoro il minatore riceve in compenso la commissione pagata da Alice.

Guardiamo con maggior dettaglio il lavoro dei *miner*, ovvero l'inserimento della transazione nella blockchain. La transazione viene aggregata insieme a diverse altre[66] in un unico blocco. Di tutte le transazioni appartenenti allo

[65] Una spiegazione con un buon equilibrio tra chiarezza espositiva e approfondimento sul funzionamento della blockchain è: https://telemaximum.com/blockchain-how-does-it-work/

[66] Un blocco della blockchain contiene centinaia di transazioni, in media 500.

stesso blocco viene calcolata la *Merkle root*[67] che sarà inclusa nell'intestazione del blocco insieme all'hash del blocco precedente, ad uno time stamp e alla risposta alla sfida computazione **proof of work** detta nonce come si può notare nel dettaglio della figura 20.

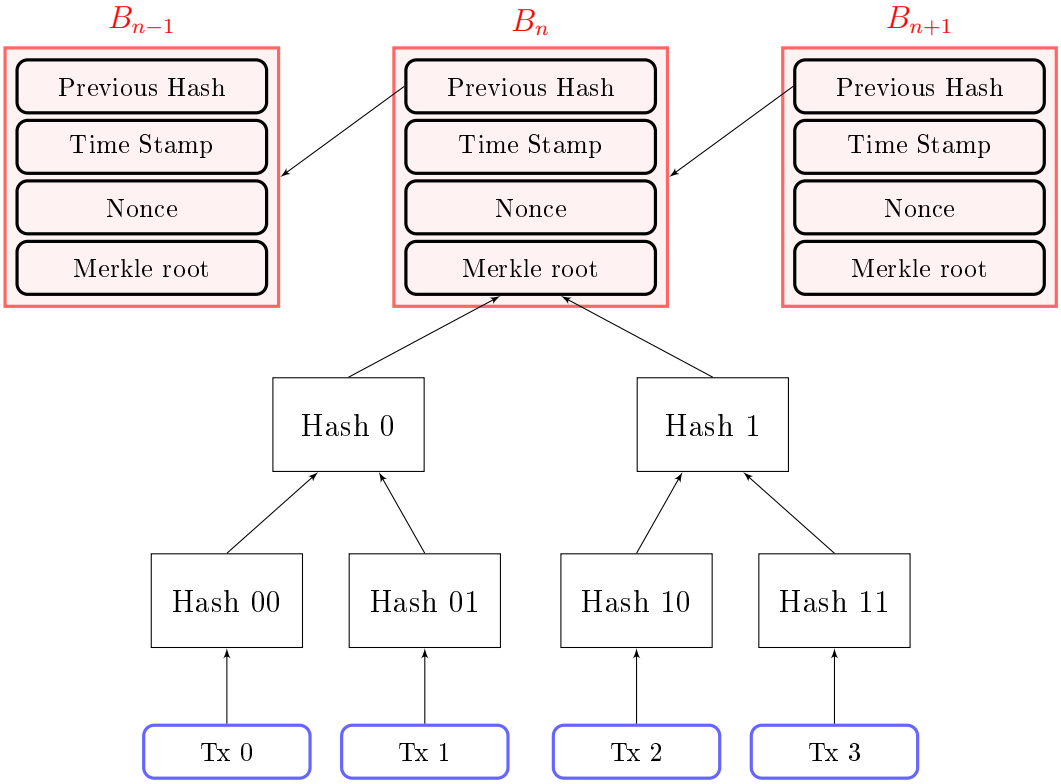


Figura 20: dettaglio di un nodo della blockchain

Si noti che la concatenazione tra i blocchi è "a ritroso": ogni nodo contiene il riferimento al precedente. Ma cos'è questo nonce e a cosa serve?
Inserire un nuovo blocco in una struttura condivisa, all'interno di una rete paritetica presenta almeno due problemi fondamentali: decidere *chi* lo inserisce e *come* gli altri nodi possono verificare e confermare l'aggiunta. In una struttura gerarchica questi due problemi si risolvono affidando il compito di mantenere aggiornato il libro mastro ad un ente centrale a cui si concede la

[67] Un Merkle Tree è un albero binario di hash dove le foglie contengono il message digest dei dati (le transazioni nel caso del bitcoin) e i nodi intermedi contengono la concatenazione degli hash dei nodi intermedi, fino ad arrivare alla radice. Si tratta di una estensione del singolo message digest: una struttura particolarmente utile per verificare i dati aggregati.

fiducia di tutta la rete, ma se si vuole fare a meno di tale ente, è necessario un meccanismo di consenso condiviso (**consensus mechanism**).

L'idea sfruttata da Satoshi Nakamoto è quella di far partecipare tutti i nodi ad una competizione, una sfida computazionale: il vincitore ottiene il diritto ad aggiungere il blocco ed il corrispondente compenso che consiste nella somma delle commissioni (fees) delle singole transazioni contenute nel blocco e da uno specifico premio detto *Coinbase transaction*[68]. Nel caso particolare la sfida consiste nel trovare il numero binario **nonce** (ovvero **n**umber used only **once**) che, concatenato agli altri campi dell'intestazione del blocco (versione, prev_hash, timestamp, merkel_root) e dato in input ad una funzione di hash, produce un message digest inferiore o uguale ad una determinata soglia (*threshold*). Per le proprietà delle funzioni di hash tale problema matematico pò essere affrontato solo con un approccio brute-force, pertanto trovare la risposta, in nonce, equivale a provare di aver investito un certo sforzo computazionale, ovvero aver dato una **proof of work**[69].

Mentre per il minatore (detto anche *prover*) lo sforzo computazionale è notevole (ed inversamente proporzionale alla soglia), per tutti gli altri nodi (*verifiers*) la verifica è computazionalmente irrilevante: il calcolo di un hash e permette a tutti i nodi della rete di accettare come valido un blocco appena minato. Questo meccanismo permette anche di risolvere il problema della spesa multipla dello stesso valore (*double spending problem*): si vuole evitare che qualcuno usi lo stesso bitcoin per pagare più persone, ovvero inserisca lo stesso output come input in più transazioni. Se ciò accade in momenti diversi, la seconda transazione viene scartata in quanto è facile verificare il duplicarsi della coppia di valori (Hash key e Index Number). se invece due transazioni contenenti lo stesso input fossero inviate e minate contemporaneamente da nodi indipendenti si verificherebbe una biforcazione nella blockchain. In tal caso basta aspettare la concatenazione dei nodi successivi: solo la catena più lunga è considerata valida.
La probabilità che continuino ad essere minati altri blocchi simultaneamente in successione diventa praticamente nulla dopo 6 blocchi: questo è il numero di blocchi successivi che si deve aspettare per essere sicuri della validità di una transazione.

[68]Una transazione Coinbase è una transazione specifica che può essere creata solo da un miner; non ha input ed è generata ogni volta che un nuovo blocco è minato, ovvero aggiunto alla blockchain. Il valore della Coinbase transaction si dimezza ogni 210 mila blocchi. Inizialmente era pari a 50 BTC e l'ultimo dimezzamento (il terzo) è avvenuto nel maggio del 2020, per tanto il premio attuale per i minatori è di 6,25 BTC a blocco.

[69]L'idea originale della proof of work è stata sviluppata nel 1997 da Adam Back, crittografo e cyberpunk inglese, in un sistema detto Hashcash, inventato per limitare lo spamming e il DoS attack

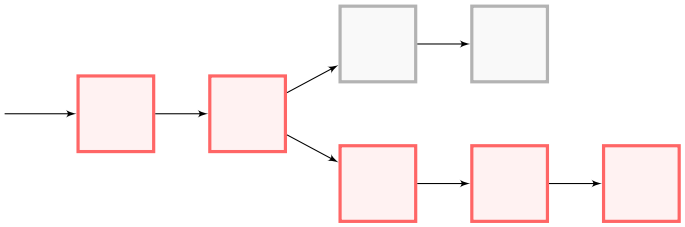


Figura 21: biforcazioni nella blockchain. Vince la catena più lunga.

Come abbiamo già affermato, trovare il **nonce** richiede una grande potenza di calcolo. Se provassimo a minare un blocco con il nostro PC impiegheremmo probabilmente diversi mesi[70]. Questo tipo di calcolo si presta ad essere parallelizzato e svolto da pool di computer e sono stati creati processori appositi per tale lavoro: mining application-specific integrated circuit (ASICs). Nel caso queste risorse venissero accentrate in misura del 51% verrebbe meno i meccanismo del consenso distribuito.

C'è però un problema più urgente rispetto al cosiddetto *51% attack* ed è il consumo di energia elettrica richiesto dal mining: esso ha raggiunto nel 2020 livelli insostenibili[71].

Una alternativa alla proof of work è la **proof of stack**, traducibile in "prova che si ha un interesse in gioco", usata ad esempio dalla cripto-valuta Ethereum. L'interesse di un nodo può essere misurato su parametri quali il valore posseduto che si è disposti a congelare e il tempo di per il quale si è lasciato congelato. Il confronto tra queste cripto-valute e i loro meccanismi di consenso distribuito va oltre gli scopi di queste note. Ci limiteremo a notare che la tecnologia blockchain ha e avrà un ruolo molto rilevante nella vita economica, produttiva e sociale a livello planetario, quindi conoscerne i meccanismi ha un valore che va oltre l'interesse specialistico dello studente informatico. A livello tecnico, è interessante approfondire la piattaforma di tecnologia blockchain per qualsiasi tipo di applicazione distribuita *EOSIO* https://eos.io/ sviluppata in C++ con licenza open source.

[70] https://coindoo.com/how-long-will-it-take-to-mine-1-bitcoin-on-your-pc/

[71] Per un articolo sui costi energetici del mining vedi *The Bitcoin Network Now Consumes 7 Nuclear Plants Worth of Power* https://news.bitcoin.com/the-bitcoin-network-now-consumes-7-nuclear-plants-worth-of-power/

15 Crittografia quantistica

La crittografia quantistica sfrutta alcune delle caratteristiche 'paradossali' o intrinsecamente inconoscibili della fisica delle particelle per risolvere il problema dello scambio delle chiavi o per condividere un *one time pad*, ottenendo così un sistema crittografico matematicamente sicuro. I principi della meccanica quantistica sono 'strani', contro-intuitivi e paradossali tanto che lo stesso Niels Bohr[72], uno dei padri di tale teoria, ebbe a dire che

> "chiunque possa contemplare la meccanica quantistica senza rimanerne frastornato non l'ha capita".

Per una introduzione sistematica, seppur accessibile, alla fisica dei quanti si rimanda agli appunti del prof. Giulio Nardon[73]. In queste note riprenderemo solo i due concetti alternativi di *sovrapposizione degli stati* e di *multiverso*. Il principio di sovrapposizione degli stati afferma che un sistema quantistico assume *contemporaneamente* tutti gli stati possibili (ad esempio la polarizzazione lungo un'asse) fino a che non interviene una osservazione, ovvero un procedimento di misura che lo costringe ad assumere uno solo di tali stati.

Shroedinger[74] formulò il famoso paradosso del gatto per tentare di chiarire questi concetti.

Un esempio importante per la trasmissione dati è dato dal comportamento dei fotoni. Un fotone polarizzato lungo la bisettrice di un sistema di rilevazione assume sia la polarizzazione verticale che quella orizzontale. All'atto della rilevazione, un filtro verticale ha la probabilità del 50% di farlo passare e lo stesso vale per un filtro orizzontale. Una visione alternativa è quella degli universi paralleli: prima di essere osservato il fotone esiste in almeno due universi paralleli, in uno di essi è polarizzato lungo la verticale e nell'altro è polarizzato lungo l'orizzontale. Solo il processo di misura costringe il fotone in uno solo degli universi possibili e abbiamo il 50% di probabilità di scegliere quello giusto. Va inoltre notato che se scegliamo un filtro di polarizzazione identico a quello con cui il fotone è stato inizialmente polarizzato (nel caso in esempio orientato a 45 gradi rispetto l'orizzonte) abbiamo il 100% di probabilità di farlo passare e quindi di rilevarlo.

[72]Niels Henrik David Bohr (Copenaghen, 7 ottobre 1885 – Copenaghen, 18 novembre 1962) è stato un fisico danese che propose il primo modello quantistico dell'atomo di idrogeno per il quale vinse il premio Nobel nel 1922. tra i suoi allievi più illustri ci fu Heisenberg. Sulla porta del suo studio teneva il simbolo del tao a simboleggiare la dualità dei fenomeni

[73]qui va messo il link agli appunti sulla teoria quantistica.

[74]E.Shroedinger Vienna 18.. - 19.. Premio Nobel per la fisica nel 19.. per la sua equazione d'onda

Questa strana natura del comportamento dei fotoni può essere usata per condividere una chiave o un one time pad. Sono necessari:

- una fonte di singoli fotoni polarizzati
- un set di filtri polarizzanti ortogonali lungo le direzioni verticale-orizzontale (che indicheremo don + o VH)
- un set di filtri polarizzanti ortogonali lungo le direzioni diagonali (che indicheremo con x o LR)
- un canale che trasporti fotoni, ad esempio una fibra ottica
- un rilevatore di singoli fotoni

Ricordiamo anche che in base ad uno dei postulati fondamentali della meccanica quantistica è impossibile conoscere i dettagli di un sistema senza perturbarlo[75] e tale caratteristica può essere usata per verificare se la trasmissione è stata intercettata da Eve. Se Eve vuole intercettare i fotoni scambiati tra Alice e Bob ne perturba lo stato e, da un controllo a posteriori, Alice e Bob possono rilevare tale interferenza.

La procedura richiede 5 passi:

1. Alice spedisce a Bob una serie di fotoni (scegliendo casualmente tra le quattro possibili polarizzazioni) e Bob li "misura", ovvero ne stima la polarizzazione scegliendo casualmente i filtri + oppure x
2. Alice e Bob comunicano vicendevolmente (rendendo quindi pubbliche) le rispettive polarizzazioni scelte. In questo modo possono sapere quali misure di Bob sono corrette
3. Alice e Bob scartano i bit per i quali Bob non ha scelto la stessa polarizzazione di Alice (ciò accade circa il 50% delle volte) e tengono quelli in cui la misura è corretta così da condividere due chiavi identiche
4. Alice a Bob controllano l'integrità della chiave appena condivisa inviandosi alcuni bit (ad esempio 80 bit su 1000) che, in quanto pubblici, saranno quindi eliminati dalla chiave.
5. Se il controllo è soddisfacente, significa che la comunicazione non è stata falsata da Eve quindi Alice e Bob usano la chiave per cifrare il messaggio; se si riscontra un errore nei bit di controllo, alice e Bob ricominciano la procedura dal punto 1

	1	2	3	4	5	6	7	8	9	10	11
A	⤢	↔	↕	⤡	↔	⤡	↕	⤡	⤢	↔	↕
B	+	+	×	×	+	+	×	+	×	×	+
	✗	✓	✗	✓	✓	✗	✗	✗	✓	✗	✓

Tabella 3: A: fotoni spediti da Alice; B: filtri scelti da Bob

Questo procedimento è stato ideato nel 1984 da Charles Bennet[76] e Gilles Brassard e da loro ha preso il nome di **BB84**. La prima realizzazione pratica fu condotta nel 1989 dallo stesso Bennet, mediante apparati non particolarmente sofisticati da lui stesso assemblati. Con l'aiuto di poche persone riuscì a far condividere una chiave tra Alice e Bob, due computer distanti 40 cm uno dall'altro. Questo primo esperimento ebbe il merito di abbattere lo scetticismo di una buona parte della comunità scientifica che, se da una parte riconosceva il valore teorico della crittografia quantistica, dall'altro nutriva forti dubbi che potesse avere una implementazione pratica, soprattutto per la difficoltà di isolare i singoli fotoni. Tra le realizzazioni più importanti va ricordata la **DARPA Quantum Network**, la prima rete a distribuzione di chiave quantistica diventata operativa nel 2003[77]. Di notevole importanza è anche l'esperimento che costituisce il passo iniziale per l' **Italian quantum backbone** (Iqb), una rete di comunicazione quantistica capace di garantire la privacy degli utenti e la sicurezza dei dati. L'esperimento è stato condotto con successo nel dicembre del 2019 dai ricercatori del CNR-INO e del LENS di Firenze, in collaborazione con l'Università Tecnica della Danimarca, che hanno testato nell'area di Firenze un sistema di comunicazione quantistica, sfruttando come canale di trasmissione una porzione della dorsale italiana in fibra ottica, una rete di circa 1.800 km realizzata dall'INRiM, che collega l'Italia da Torino a Matera[78].

[75]Da questo postulato si può, ad esempio, ricavare il principio di indeterminazione di Heisenberg.

[76]C. Bennet mise a punto il suo sistema mentre era Reseach fellow presso i laboratori IBM ispirato dall'idea di "quantum money" di un suo vecchio amico, Stephen Wiesner.

[77]https://en.wikipedia.org/wiki/DARPA_Quantum_Network

[78]https://www.cnr.it/it/comunicato-stampa/9115/primo-test-italiano-di-crittografia-quantistica

15.1 Il computer quantistico

La crittografia quantistica non ha ancora preso piede ma potrebbe diventare essenziale quando il computer quantistico diventerà una tecnologia matura. Il computer quantistico fu delineato nella sua struttura teorica nel 1984 da David Deutsch[79] nell'articolo *Quantum theory, the Church-Turing principle and the universal quantum computer*[80] dove, partendo dall'osservazione che i computer tradizionali si basano sulla fisica classica, introduce il concetto di bit quantistico o **qbit** come un elemento fondamentale di informazione che, a differenza del bit, può assumere "contemporaneamente" i valori 1 e 0, come sovrapposizione di stati di una particella subatomica. Un qbyte, quindi, non conterrebbe solo una delle 256 possibili combinazioni, ma le contiene tutte contemporaneamente come sovrapposizione di tutti gli stati possibili e, se si riesce a scrivere un algoritmo in grado di estrarre[81] la combinazione voluta, si può ridurre drasticamente la complessità computazionale di alcuni problemi. Tra i primi algoritmi scritti per un computer quantistico troviamo proprio quello della fattorizzazione di un numero. L'autore, Peter Shor[82], nel celebre articolo del 1994 *Algorithms for quantum computation: discrete logarithms and factoring*[83] dimostra che, almeno in via teorica, è possibile con un computer quantistico fattorizzare un numero di n bit in un tempo polinomiale anziché esponenziale come succede con un computer classico. Le basi fisiche e matematiche necessarie per analizzare l'algoritmo di Shor vanno oltre le conoscenze degli studenti cui è rivolto questo scritto (e, per la verità, anche di quelle di chi scrive) ma la sua validità è stata sperimentalmente dimostrata sul *IBM Q System One*, un computer quantistico a 20 qbit. E' di pochi

[79] (Haifa - Israele, 18 maggio 1953) è un fisico britannico. Premiato con il premio Dirac nel 1998, è professore presso il dipartimento di fisica atomica e laser presso il centro per la computazione quantistica, nel laboratorio Clarendon dell'Università di Oxford [wikipedia] . Va ricordato che tra il nostro istituto ed il padre del computer quantistico c'è un solo grado di separazione: possiamo infatti vantarci di essere amici, grazie a *la Via delle Scienze* di Chiara Marletto, ricercatrice in fisica quantistica all'università di Oxford dove collabora con David Deutch alla constructor theory http://laviadellescienze.altervista.org/marletto/

[80] https://www.jstor.org/stable/2397601?seq=1text

[81] L'estrazione di uno dei possibili stati in cui si trova una serie di qbit equivale ad una *misurazione*: nella meccanica quantistica la misura fa collassare il sistema in uno solo dei possibili stati. Ritornando al gatto di Shroedinger, quando apriamo la scatola (e lo guardiamo), esso si troverà in uno solo dei due stati possibili: vivo o morto. Dal punto di vista matematico si dice che la misurazione di un singolo qbit proietta lo stato quantico su uno degli stati della base (vettoriale complessa) associati con il dispositivo di misurazione.

[82] Peter Williston Shor (New York, 14 agosto 1959) è attualmente docente di matematica applicata presso il MIT di Boston. Ha sviluppato il suo famoso algoritmo di fattorizzazione nel 1994, mentre lavorava presso i Bell Labs.

[83] https://www.computer.org/csdl/proceedings-article/focs/1994/0365700/12OmNqNXErh

mesi fa, inoltre, la notizia che Sycamore, il computer quantistico a 53 qbit sviluppato ga Google ha risolto in 200 secondi un calcolo che un supercomputer tradizionale risolverebbe in 10.000 anni [84]. Se sarà tecnologicamente possibile rendere stabili[85] computer quantistici con un alto numero di qbit, l'infrastruttura crittografica su cui attualmente si basa la sicurezza delle comunicazioni sul web ed altre tecnologie come le cripto-valute verranno meno. A quel punto, probabilmente, la contromossa si potrà trovare ancora nella meccanica quantistica, come descritto nel paragrafo precedente.

[84] Why Google's Quantum Supremacy Milestone Matters: https://www.nytimes.com/2019/10/30/opinion/google-quantum-computer-sycamore.html

[85] Uno dei problemi tecnologici da risolvere per i computer quantistici è quello della perdita di **coerenza**, ovvero il collasso della sovrapposizione di stati dovuto all'interazione con l'ambiente esterno. Per un approfondimento su tali aspetti tecnici e fisici si veda https://cordis.europa.eu/article/id/418227-maintained-coherence-for-better-quantum-information-applications/it

A Appendice: Generazione chiavi RSA

Viene riportato di seguito il codice C per la generazione delle chiavi e per la cifratura e decifratura dei messaggi. È un codice ottimizzato per sistemi linux a 64 bit, ma può essere compilato anche in sistemi Windows.

```
/* Luca Battistin - 2013  - rev feb 2017 - GNU GPL */
/*********************************************************
  Algoritmo Gnerazione Chiavi RSA - a soli scopi didattici
  con lunghezza inferiore a 16 cifre decimali
  OTTIMIZZATO per Linux x86 a 64 bit
  N.B. compilare con l'opzione -lm
  gcc -o rsa_keygen.out rsa_key_gen.c -lm
*********************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#define false 0
#define true 1

typedef unsigned long int lungo64;

int is_prime(lungo64 x);
lungo64 next_prime(lungo64 x);
lungo64 MCDe(lungo64 a, lungo64 b, int* step);
lungo64 EuclideEsteso(lungo64 a, lungo64 b, int n);
lungo64 LimitiRandom(int num_cifre);

int main(int argc, char **argv)
{
    int l_chiavi;
    lungo64 n,p,q,z,e,mcd,d;
    lungo64 M;
    int passi;
    int l_max, ll_max;   //lunghezze massime delle chiavi (a 32 o 64 bit)
    char risposta[255];
    printf("\n************** Generazione chiavi RSA ****************\n");
    printf("** Algoritmo semplificato per capirne il funzionamento **\n");
    if(sizeof(lungo64) == 4){
        printf("-----------------------------------------------\n");
```

```
36          printf("-         Attenzione: sistema a 32 bit          -\n");
37          printf("-   Dimensione massima delle chiavi : 4 cifre   -\n");
38          printf("-  Con chiavi più lunghe si rischia l'overflow  -\n");
39          printf("-      (ottimizzato per sistemi a 64 bit)       -\n");
40          printf("-------------------------------------------------\n");
41          printf("Scegli la lunghezza delle chiavi (min 2 - max 4)\n");
42          l_max = 4;
43          ll_max = 9;
44      }
45      else{
46          printf("Scegli la lunghezza delle chiavi (min 2 - max 8)\n");
47          l_max = 8;
48          ll_max = 18;
49      }
50      scanf("%d",&l_chiavi);
51      if (l_chiavi < 2 || l_chiavi > ll_max){
52          printf("\nlungheza chiavi non accettabile: %d",l_chiavi);
53          printf("\n!!! Terminazione programma\n");
54          exit(1);
55      }
56      else if(l_chiavi > l_max){
57          printf("Attento!\n``)
58          printf("una chiave di %d cifre può essere calcolata\n",l_chiavi);
59          printf("Ma potrebbe provocare overflow in rsa_cipher.c\n");
60          printf("vuoi continuare? (s/n)");
61          scanf("%s", risposta);
62          if (risposta[0] != 'S' && risposta[0] != 's'){
63              printf("\n!!!! Programma terminato\n");
64              exit(1);
65          }
66      }
67      printf("lunghezza chiavi : %d\n",l_chiavi);
68      printf("Scelta dei due primi: ");
69      srand((unsigned) time(NULL));
70      M = LimitiRandom(l_chiavi/2);
71      n = rand()%M + M;
72      p = next_prime(n);
73      printf(" : %ld\t",p);
74      do {
75          n = rand()%M + M;
76          q = next_prime(n);
```

```
77      }while(q==p);    //controllo che p e q non siano uguali
78      printf(" : %ld\n",q);
79      n = p*q;
80      printf("n = (p*q)       = %ld\n",n);
81      z = (p-1)*(q-1);
82      printf("z = (p-1)*(q-1) = %ld\n",z);
83      passi=0;
84      do{
85          M = LimitiRandom(l_chiavi - 1);
86          e=next_prime(rand()%M + M); //!! da rivedere
87          printf("e: %ld\n",e); //!! debug
88          e=e%z;
89          printf("e mod z: %ld\n",e); //!! debug
90          printf("prova %ld \t ",e);
91          mcd = MCDe(z,e,&passi);
92          printf("MCD (%ld,%ld) = %ld",e,z,mcd);
93      }while(mcd!=1);
94      printf("\n********* scelta di e ********* \n ");
95      printf("%ld \n",e);
96      printf("******** Calcolo di d **********\n");
97      printf("passi di euclide : %d\n", passi);
98      d=EuclideEsteso(z,e,passi);
99      printf("d = %ld\n\n",d);
100     /* scrittura chiavi sui file */
101     FILE* fout=fopen("RSA_K_pri.txt","wt");
102     if(fout==NULL){
103         printf("Errore in apertura del file RSA_K_pri.txt\n");
104         exit(1);
105     }
106     fprintf(fout, "%ld %ld",d,n);
107     fclose(fout);
108     fout=fopen("RSA_K_pubb.txt","wt");
109     if(fout==NULL){
110         printf("Errore in apertura del file RSA_K_pubb.txt\n");
111         exit(1);
112     }
113     fprintf(fout, "%ld %ld",e,n);
114     fclose(fout);
115     printf("Chiavi salvate nei file RSA_K_pubb.txt e RSA_K_pri.txt\n\n");
116     return 0;
117 }
```

```
118 /* Funzione che verifica se il numero x è primo
119  * E' molto inefficace, ma segue il metodo più ovvio.
120  * Molto potrebbe essere migliorata con il crivello di Eratostene */
121
122 int is_prime(lungo64 x)
123 {
124     if (x < 2)
125         return false;
126     lungo64 i;
127     for (i = 2; i <= sqrt(x); ++i)
128     {
129         if (x % i == 0)
130         return false;
131     }
132     return true;
133 }
134
135 /* Funzione che ritorna il prossimo numero primo maggiore di x */
136 lungo64 next_prime(lungo64 x)
137 {
138     if (x == 2) return 3;
139     if (x%2 == 0) x++;
140     for (; !is_prime(x); x+=2);
141     return x;
142 }
143
144 lungo64 MCDe(lungo64 a, lungo64 b, int* step)
145 {
146     lungo64 tmp;
147     if(b > a){
148         tmp = a;
149         a = b;
150         b = tmp;
151     }
152     while(b > 0){
153         (*step)++;
154         tmp = a;
155         a = b;
156         b = tmp % a;
157     }
158     return a;
```

```
159 }
160
161 /* Euclide Esteso per trovare il moltiplicativo inverso modulo n*/
162 lungo64 EuclideEsteso(lungo64 a, lungo64 b, int n){
163 /* Calcolo dei quozienti*/
164     lungo64 R[n+1],Q[n+1];
165     R[0]=b; R[1]=a%b;
166     Q[0]=b; Q[1]=a/b;
167     int i=1;
168     while(R[i]!=0){
169         i++;
170         R[i]=R[i-2] % R[i-1];
171         Q[i]=R[i-2] / R[i-1];
172     }
173     /* Calcolo dei coefficienti dell'equazione ax + by = MCD(a,b)*/
174     lungo64 X[n+1],Y[n+1];
175     X[0]=0; X[1]=1;
176     Y[0]=1; Y[1]=0;
177     for(i=2;i<=n;i++){
178         X[i]=X[i-2] - X[i-1]*Q[i-1];
179         Y[i]=Y[i-2] - Y[i-1]*Q[i-1];
180     }
181     X[n] = (X[n] + a) % a;
182
183     return X[n];
184 }
185
186 /* Ritorna 5 con tanti zeri quanti indicati da num_cifre -1
187  * serve per limitare la funzione rand() */
188 lungo64 LimitiRandom(int num_cifre){
189         lungo64 M = 5;
190         int j;
191         for (j=1; j< num_cifre; j++)
192             M = M * 10;
193         return M;
194     }
```

B Appendice: cifratura RSA

```c
/*****************************************************************/
/*     ITI Marzotto di Valdagno    ***    www.iisvaldagno.it     */
/*     prof. Luca Battistin                                      */
/*     Lezioni di Crittografia           2016-2017               */
/*     Prova di cifratura RSA                                    */
/*     sorgente rilasciato con licenza GNU GPL                   */
/*     Disponibile nel corso  www.u-learning.it                  */
/*****************************************************************/
/* Le chiavi pub(e,n) e pri(d,n) sono state generate da
 * rsa_key_gen.c    */
/* Il (semplicissimo) programma non esegue altro che (base ^ e) mod n
 * può essere usato sia per cifrare che per decifrare */
/* N.B. per i limiti del tipo long a 64 bit, la lunghezza delle chiavi
 * dovrebbe rimanere inferiorealle 10 cifre decimali */
#include <stdio.h>
#include <stdlib.h>
/* Funzione che cifra: a^b mod n*/
typedef unsigned long int lungo64;
lungo64  rsa_cifra(lungo64  base, lungo64  esp, lungo64 modulo);
void Help(char * nomefile);

int main(int argc, char **argv){
    printf("-----------------------------------------------------\n");
    printf("- ITI Marzotto - L.Battistin -- Sistemi e Reti 5 D    -\n");
    printf("-             Prova de-cifratura RSA in C             -\n");
    printf("-           con numeri piccoli (<11 cifre)            -\n");
    printf("-         (ottimizzato per sistemi a 64 bit)          -\n");
    printf("-----------------------------------------------------\n");
    //printf("numero parametri %d\n",argc);
    lungo64 a,b,c,d;
    FILE* fin;
    int numeri_letti;
    //controllo parametri passati
    if(argc > 3){
        printf("sintassi: %s chiave_rsa file_da_cifrare\n",argv[0]);
        printf("usare %s -h per l'help", argv[0]);
        exit(1);
    }
    if(sizeof(lungo64) == 4){
```

```
40      printf("-------------------------------------------------------\n");
41      printf("-              Attenzione: sistema a 32 bit           -\n");
42      printf("-       Dimensione massima delle chiavi : 4 cifre     -\n");
43      printf("-      Con chiavi più lunghe si rischia l'overflow    -\n");
44      printf("-         (ottimizzato per sistemi a 64 bit)          -\n");
45      printf("-------------------------------------------------------\n");
46      }
47      if(argc > 1){
48          if (argv[1][0]=='-' && argv[1][1]=='h'){
49              Help(argv[0]);
50              return 0;
51          }
52          // acquisizione coppia di numeri che costituisce la chiave rsa
53          fin=fopen(argv[1],"rt");
54          if(fin==NULL){
55              printf("Errore in apertura del file %s\n",argv[2]);
56              exit(1);
57          }
58          numeri_letti = fscanf(fin, "%ld %ld", &b,&c);
59          fclose(fin);
60          if(numeri_letti != 2){
61              printf("errore nella lettura della chiave!!\n");
62              printf("programma terminato!!\n");
63              exit(1);
64          }
65      }
66      else{
67          printf("Inserisci esponente e modulo (chiave rsa): ");
68          scanf("%ld %ld",&b,&c);
69      }
70      // apertura file e acquisizione numero da cifrare
71      if(argc == 3){
72          fin=fopen(argv[2],"rt");
73          if(fin==NULL){
74              printf("Errore in apertura del file da cifrare %s\n",argv[2]);
75              exit(1);
76          }
77          numeri_letti = fscanf(fin, "%ld", &a);
78          fclose(fin);
79          if(numeri_letti != 1){
80              printf("errore nella lettura del file di input!!\n");
```

```
81              printf("programma terminato!!\n");
82              exit(1);
83          }
84      }
85      else{
86          printf("Inserisci il numero da cifrare: ");
87          scanf("%ld",&a);
88      }
89      /* Controllo consistenza dei tre numeri */
90      if(a >= c || b >= c){
91          printf ("input non accettabile: ");
92          printf("%ld e %ld devono essere minori del modulo %ld\n",a,b,c);
93          Help(argv[0]);
94          exit(1);
95      }
96      d = rsa_cifra(a,b,c);
97      printf("Usando il prodotto in algebra modulare: \n");
98      printf("%ld elevato alla %ld mod %ld: %ld\n",a,b,c,d);
99      //scrittura su file di output
100     FILE* fout=fopen("cifrato_con_RSA.txt","wt");
101     if(fout==NULL){
102         printf("Errore in apertura del file\n");
103         exit(1);
104     }
105     fprintf(fout, "%ld",d);
106     fclose(fout);
107     printf("Output salvato nel  cifrato_con_RSA.txt\n\n");
108     return 0;
109 }
110
111 void Help(char * nomefile){
112     printf("------------------------------------------------------------\n");
113     printf("- Il programma accetta come input tre numeri interi    -\n");
114     printf("- m : ovvero ciò che deve essere cifrato o decifrato   -\n");
115     printf("- e oppure d : prima parte dela chiave rsa (esponente) -\n");
116     printf("- n: seconda parte della chiave rsa (modulo)           -\n");
117     printf("- n deve essere maggiore sia di m che di e (oppure d)  -\n");
118     printf("- Tali numeri possono essere forniti da tastiera       -\n");
119     printf("- In tal caso avviare il programma senza parametri     -\n");
120     printf("- oppure possono essere forniti come file di testo:    -\n");
121     printf("- sintassi: %s chiave_rsa file_da_cifrare    -\n", nomefile);
```

```
122     printf("- usare %s -h per l'help                    -\n", nomefile);
123     printf("-------------------------------------------------------\n");
124     printf("- (evitare chiavi di lunghezza superiore alle 10 cifre -\n");
125     printf("-------------------------------------------------------\n");
126 }
127
128 lungo64  rsa_cifra(lungo64  base, lungo64  esp, lungo64 modulo){
129     unsigned long p=1;
130     int i;
131     for (i=0;i<esp;i++)
132         p=(p*base) % modulo;
133     return (unsigned int ) p;
134 }
```

C Appendice: qualche comando openssl

Openssl è una libreria di funzioni crittografiche nonché una implementazione open source dei protocolli TLS/SSL (Transport Layer Security and Security Sockets Layer).

```
### Raccolta di Comandi openssl per la crittografia.

# Per produrre il message digest del file prova.txt
openssl sha1 prova.txt > messagedigest.prova

# Per la lista completa:
# openssl list-message-digest-commands
# openssl dgst --help

#Per creare una password salted con md5
openssl passwd -1 -salt salepepe pippo
#produce in output: $1$salepepe$O4/Id8O7oiUrIK5V2iM0Z1

# per la cifratura simmetrica:
# openssl enc --help
#qualche esempio:
openssl enc -des3  -in file_da_cifrare
#oppure
echo "ciao mondo" | openssl enc -des > mess.des
# chiede la passphrase in modo interattivo
```

```
# per la decifratura ( usare l'opzione -d  "decifratura" )
openssl enc -des -d -in mess.des -out mess.des.dec

# se abbiamo salvato la password su file
openssl enc -blowfish -d -in file_cifrato -kfile passphrase_file

# Per generare una coppia di chiavi rsa:
openssl genrsa -h
openssl genrsa -out chiave.privata [1024 | 2048]
openssl rsa -pubout -in chiave.privata -out key.pub

#per cifrare il file plaintex con key.pub
openssl rsautl -h
openssl rsautl -encrypt -pubin -inkey key.pub -in plaintext > mess.cif

#per decifrare il file mess.cif con key.pri e ottenere mess.decif
openssl rsautl -decrypt -inkey key.pri -in mess.cif -out mess.decif

# Firma Digitale
# Per "firmare un documento" usare l'opzione -sign: esempio
openssl dgst -sha256 -sign key.pri -out firmaDoc NomeDoc

# Quindi per verificare autenticità e integrità:
openssl dgst -sha256 -verify key.pub -signature firmaDoc NomeDoc

#in caso di autenticità risponde
Verified OK

#Lo schema di firma completo però prevede 4 passi:
# 1) ricevo documento + firma + catena dei certificati
# 2) verifico autenticità del certificato
#    fornendo tutta la catena di certificati
openssl verify certificato_mittente.crt certificato_CA.crt ...
# 3) estraggo la chiave pubblica dal certificato
openssl x509 -pubkey -noout -in cert_mittente.crt > pubK
# 4) verifico la firma del documento:
openssl dgst -sha256 -verify pubK -signature firmaDoc NomeDoc
```

D Appendice: Breve cronologia della macchina Enigma

nel 1918 Arthur Scherbius brevetta la sua macchina cifrante elettromeccanica e le dá il nome di Enigma.

nel 1926 L'esercito ed il governo Tedesco adottano una versione più sofisticata della macchina Enigma rispetto a quella commerciale.

nel 1931 Hans Tylo Schmidt, fratello del generale (Rudolf Schmidt) responsabile della sicurezza delle comunicazioni dell'esercito tedesco, vende ai francesi alcuni documenti che permettono di capire il cablaggio interno dei rotori e quindi di costruire una copia della macchina Enigma

tra il 1932 e il 1933 Marian Rejewski, talentuoso matematico impiegato nell'ufficio di crittografia e crittanalisi polacco (Biuro Szyfrow), escogita un modo per decifrare la macchina Enigma sfruttando il fatto che la chiave del messaggio viene ripetuta due volte e costruendo le cosiddetta bomba.

agli inizi del 1939 l'esercito tedesco aggiunge due rotori e quattro cavi al pannello di commutazione (plugboard) aumentando cosí di 15 mila volte le impostazioni possibili, arrivando a 159 milioni di milioni di milioni.

il 16 agosto 1939 una replica della macchina Enigma e le conoscenze acquisite dall'intelligence polacca vengono passate agli alleati franco-inglesi.

il 4 settembre 1939 Alan Turing passa da Cambridge a Bletchley Park dove inizia il suo servizio come crittanalista.

agli inizi del 1940 Turing completa il progetto delle sue bombe: macchine ispirate dal lavoro di Rejewski, composte da 12 batterie di 3 repliche dei rotori della macchina Enigma, ingegnosamente cablati per trovare la configurazione iniziale dei rotori a partire da un *crib*.

14 marzo 1940 La prima bomba - costruita dalla *British Tabulting Machinery factory* di Letchworth seguendo il progetto di Turing viene consegnata a Bletchley Park. L'unitá completa era alta un paio di metri, altrettanto lunga e larga un metro. Purtroppo, una volta messa in funzione si rivela poco efficiente: impiega una settimana ad elaborare la chiave.

8 agosto 1940 Bletchley Park ha la prima bomba funzionante dopo che il team di Turing ha apportato delle migliorie al progetto per renderla più veloce. Nei 18 mesi successivi ne vengono costruite altre 18. Tra i perfezionamenti piú notevoli va citata la *diagonal board* di Gordon Welchman.

alla fine del 1942 ci sono 49 bombe operanti a Bletchley Park (e a Gayhurst Manor) dove vengono decifrati quasi 4 mila messaggi al giorno e dove lavorano circa 7 mila persone.

8 dicembre 1943 Tommy Flower, dopo dieci mesi di lavoro, realizza Colossus: progettato per debellare un codice ancor piú resistente di Enigma: la cifratura realizzata con la macchina Lorenz SZ40 (usata da Hitler per comunicare con i propri ufficiali). Il progettista di Colossus é Max Newman, un matematico di Bletchley che, partendo dal concetto di macchina universale di Turing, progetta un calcolatore in grado di adattarsi a problemi differenti. Colossus contiene 1500 valvole elettroniche ed é programmabile: é il precursore del computer moderno.

7 giugno 1954 Alan Turing entra nella sua camera da letto con un recipiente pieno di soluzione di cianuro e una mela. Immerge il frutto nel veleno e mette fine cosí, a quarantadue anni, alla sua vita. Rimane uno dei più grandi crittanalisti della storia e uno dei padri dell'informatica.

Riferimenti bibliografici

[1] Wade Trapp - Lawrence C. Washington: *Crittografia* con elementi di teoria dei codici Pearson, Seconda Edizione 2009

[2] Simon Singh *The Code Book* The Science of Secrecy from Ancient Egypy to Quantum Cryptography Fourth Estate Limited - London - 1999

[3] Andrew Hodges *Alan turing. Storia di un Enigma* La Storia vera di una mente straordinaria Bollati Boringhieri, 2014

[4] https://www.khanacademy.org/computing/computer-science/cryptography/modern-crypt/v/diffie-hellman-key-exchange-part-1 *Journey into cryptography: Diffie Hellman Key Exchange* The Khan Academy web site, 2013

[5] http://infocom.uniroma1.it/alef/labints/Text/sicurezza.html *Dispositivi e Meccanismi di Sicurezza* Alessandro Falaschi, Dipartimento di Inge-

gneria dell'Informazione, Elettronica e Telecomunicazioni dell'Università di Roma Sapienza

[6] Bruce schneier *Data and Goliath* The hidden battles to collect your data and control your world. W.W. Norton & Company Inc. - New York - 2015

[7] http://infocom.uniroma1.it/alef/labints/Text/sicurezza.html *Tor: The Second-Generation Onion Router* Roger Dingledine, Nick Mathewson, Paul Syverson. Articolo del 2004

[8] http://satoshinakamoto.me/bitcoin.pdf *Bitcoin: A Peer-to-Peer Electronic Cash System* Satoshi Nakamoto. www.bitcoin.org. 2009