Liceo cantonale Bellinzona

Liceo Cantonale di Bellinzona

LAVORO DI MATURITÀ IN MATEMATICA

Crittografia: dagli albori al XXI secolo

L'eterno compromesso tra sicurezza e usabilità

Candidato:
Niccolò Cavallini

Docente:
Prof.ssa Alessandra Rigato

Anno 2020

Indice

Più forziamo i confini della matematica, più questi si allargano. Non correremo mai il rischio di esaurire i nuovi problemi da risolvere.

— Ian N. STEWART, 1945 – *, matematico e scrittore britannico

Introduzione

Questo Lavoro di Maturità analizza l'evoluzione della Crittografia, disciplina che accompagna l'Uomo da millenni, nata dalla necessità di secretare informazioni. Il testo introduce anzitutto alcune basi matematiche necessarie al lettore per comprendere a fondo il funzionamento dei vari algoritmi crittografici esposti. Tali prerequisiti toccano argomenti quali la Teoria dei Gruppi e l'Aritmetica modulare.

La presentazione dei diversi algoritmi crittografici segue un filo cronologico: tratterò inizialmente le tecniche conosciute fin da prima di Cristo, per poi delineare i tratti che la crittografia possedeva nel XX secolo ed addentrarmi infine negli algoritmi che vengono utilizzati in questa nostra era digitale. Ed è proprio l'Informatica, oltre che il rinnovato interesse che i matematici e i crittografi hanno avuto nel Novecento per la Teoria dei Numeri, ad aver permesso alla Crittografia uno sviluppo impressionante, tanto che gli algoritmi precedenti appaiono oggi visibilmente rudimentali. In effetti, la Teoria dei Numeri e la Crittografia hanno posto le basi e creato il pretesto per affrontare studi scaturiti nell'indagine di un problema conosciuto, in realtà, già dagli Antichi Greci, ossia quello di determinare la primalità di un numero naturale e di ricavarne i fattori primi. Con la potenza di calcolo senza precedenti offerta dai computer, questo problema non è apparso meno interessante; al contrario, ha acquisito importanza, proprio perché oggi se ne possono studiare gli aspetti più profondi. Ed è proprio il fatto che, ad oggi, non esista un algoritmo computazionalmente efficiente a sufficienza per determinare velocemente la scomposizione in fattori di un intero di centinaia di cifre, a garantire il corretto funzionamento di molti crittosistemi, penso in particolare all'RSA. Questo fatto è talmente fondamentale che il problema sopra descritto è spesso chiamato *Problema RSA*. Al momento in cui questo problema avrà una soluzione efficiente, ci troveremo a dover ripensare, se non l'avremo già fatto, i nostri metodi crittografici. Il Lavoro espone quindi certamente le basi matematiche necessarie all'analisi dei crittosistemi, ma ciò che adduce particolarità a questa ricerca è che quest'ultima è coadiuvata da esempi pratici che prendono la forma di codice informatico, scritto nei linguaggi Java e Python 3, che il lettore può eseguire sul suo computer per meglio comprendere il funzionamento di questi algoritmi. Per eseguire il codice, il lettore può seguire le istruzioni dettagliate fornite nell'Appendice.

Si potrà inoltre comprendere il *compromesso tra sicurezza e usabilità* a cui faccio riferimento nel sottotitolo. L'evoluzione della Crittografia è sempre stata caratterizzata dalla difficile ricerca di questo compromesso: dobbiamo poter trasmettere le informazioni in modo sicuro in un tempo ragionevolmente contenuto e con un costo, in termini di calcolo, altrettanto limitato. Noteremo come vari crittosistemi moderni siano stati creati sulla base di requisiti diversi, in funzione dell'applicazione che essi devono avere.

Il testo porterà il lettore a comprendere quanto importante sia la Crittografia nel mondo

di oggi, e quanto questa, pur basandosi su solide considerazioni matematiche, si debba adattare al contesto della quotidianità contemporanea, in cui, oltre che la sicurezza, è anche la velocità nello scambio delle informazioni a giocare un ruolo chiave.

In conclusione, desidero ringraziare la Professoressa Alessandra Rigato per gli impulsi indispensabili alla scelta del tema del presente Lavoro, per la costante rilettura e correzione delle bozze e per i suggerimenti forniti.

1 Concetti fondamentali

In questo capitolo vengono poste le basi matematiche che serviranno al lettore per comprendere a fondo i concetti di crittografia che verranno esposti in questo lavoro di ricerca. Mi sembra opportuno iniziare con alcune **definizioni**, per poi procedere con alcuni approfondimenti sull'**aritmetica modulare** e sulla **sicurezza di Shannon**.

1.1 Definizioni generali

Definizione 1.1 (Cardinalità di un insieme)**.**
Si definisce **cardinalità di un insieme** finito A il numero di elementi contenuti in A. È indicata con $\text{card}(A)$, $|A|$ o con $\#(A)$. In questo testo si preferisce la prima notazione.

Definizione 1.2 (Alfabeto)**.**
Si dice **alfabeto** un insieme finito arbitrario di simboli. È indicato con $\mathfrak{A}$.

Definizione 1.3 (Messaggio in chiaro)**.**
Si definisce **messaggio in chiaro** una qualsiasi sequenza di caratteri m che non è stata sottoposta a crittografia. L'insieme dei messaggi in chiaro è matematicamente indicato con $\mathfrak{A}^n$, dove n è il numero massimo di simboli utilizzati.

Esempio. Una parola italiana è un elemento di $\mathfrak{A}^{21}$.

Definizione 1.4 (Messaggio cifrato)**.**
Si definisce **messaggio cifrato** una sequenza di caratteri c che non è comprensibile a tutti. L'insieme dei messaggi cifrati si indica con $\mathfrak{C}$. Come per $\mathfrak{M}$, vale $\mathfrak{C} = \mathfrak{A}^n$.

Definizione 1.5 (Funzione di cifratura)**.**
Si dice **funzione di cifratura** una qualsiasi funzione f così definita:

$$\begin{aligned} f &: \mathfrak{M} \to \mathfrak{C} \\ m &\mapsto f(m) := c \end{aligned}$$

La funzione di cifratura è **biiettiva**, ossia è iniettiva e suriettiva, in modo da poterla invertire.

Definizione 1.6 (Funzione di decifratura)**.**
In modo analogo alla funzione di cifratura, la **funzione di decifratura** d è definita come l'inversa della prima. In simboli:

$$\begin{aligned} d &: \mathfrak{C} \to \mathfrak{M} \\ c &\mapsto m := d(c) := f^{-1}(c) \end{aligned}$$

Definizione 1.7 (Chiave e insieme delle chiavi)**.**
Si dice **chiave** (di cifratura) un qualsiasi oggetto k, sia esso una sequenza di caratteri, di bit, una matrice, ecc. che permette la crittazione e la decrittazione di un messaggio. Le chiavi sono gli elementi dell'**insieme delle chiavi** indicato con $\mathfrak{K}$.

Definizione 1.8 (Crittosistema)**.**
Si dice **crittosistema** la quaterna

$$(\mathfrak{M}, \mathfrak{C}, f, f^{-1})$$

Notazione. Alcuni testi includono nel concetto di crittosistema anche l'insieme $\mathfrak{K}$.

Da ultimo una definizione di carattere informatico, che ci servirà in seguito per discutere del crittosistema AES.

Definizione 1.9 (XOR)**.**
L'operazione XOR, anche detta «*or esclusivo*» e indicata dal simbolo $\oplus$, è un operazione tra bit così definita:

$$\{0,1\} \to \{0,1\}$$

$$0 \oplus 0 = 0; \quad 0 \oplus 1 = 1; \quad 1 \oplus 0 = 1; \quad 1 \oplus 1 = 0.$$

Dati due bit b_1, b_2, XOR agisce così: $b_1 \oplus b_2 = b_1 + b_2 \pmod 2$.

1.2 Teoria dei Gruppi, anelli e campi

La Teoria dei Gruppi si basa, come si può evincere dal nome, sul concetto di **gruppo**, una delle più semplici strutture algebriche possibili. Si ha un insieme e un'operazione fra gli elementi di questo insieme, alla quale si richiede di avere alcune particolari proprietà.

Definizione 1.10 (Gruppo)**.**
Un insieme G dotato di un'operazione $\diamond$ si dice **gruppo**, e si indica con $(G, \diamond)$, se sono verificate le seguenti proprietà:

(i) G è **chiuso** rispetto a $\diamond$, ovvero $\forall a, b \in G$ vale $a \diamond b \in G$;

(ii) Esiste l'**elemento neutro** $e \in G$ tale che $\forall a \in G$ vale $a \diamond e = e \diamond a = a$;

(iii) Esiste l'**inverso** $a^{-1} \in G$ $\forall a \in G$ tale che $a \diamond a^{-1} = a^{-1} \diamond a = e$;

(iv) È valida la **proprietà associativa**, ovvero $a \diamond (b \diamond c) = (a \diamond b) \diamond c$ $\forall a, b, c \in G$.

Se è valida anche la

(v) **Proprietà commutativa**, ossia $a \diamond b = b \diamond a$ $\forall a, b \in G$, allora il gruppo si dice **abeliano**.

Per fare degli esempi, possiamo ricordare che $\mathbb{Z}, \mathbb{Q}, \mathbb{R}, \mathbb{C}$ sono gruppi con l'addizione e $\mathbb{Q}^*, \mathbb{R}^*, \mathbb{C}^*$ sono gruppi con la moltiplicazione. Parlando di aritmetica modulare, $\mathbb{Z}_n$ è un gruppo con la somma modulo n e $\mathbb{Z}_n^*$ con la moltiplicazione modulo n. Va ricordato il significato di «$*$»: questo simbolo indica che dall'insieme a cui è apposto sono stati rimossi *tutti gli elementi che non sono invertibili*, i.e. tutti gli elementi per cui non si può trovare un elemento che, operato con l'originario, dia l'elemento neutro.

Definizione 1.11 (Sottogruppo)**.**
Se G è un gruppo rispetto a un'operazione $\diamond$ e $H \subset G$ è a sua volta un gruppo rispetto alla stessa operazione, allora H è detto **sottogruppo** di G.

Definizione 1.12 (Sottogruppo generato da un elemento)**.**
Dato un elemento g di un gruppo moltiplicativo G l'insieme

$$\langle g \rangle := \{e\} \cup \{g^n : n \in \mathbb{N}\} \cup \{g^{-n} : n \in \mathbb{N}\} = \{g^n : n \in \mathbb{Z}\}$$

si dice **sottogruppo generato da** g ed è un *gruppo abeliano* rispetto a $\diamond$.

Definizione 1.13 (Omomorfismo e isomorfismo tra gruppi)**.**
Due gruppi $(G, \diamond)$ e $(H, \odot)$ si dicono **omomorfi** se è possibile trovare una funzione $f : G \to H$ tale che

$$f(a \diamond b) = f(a) \odot f(b) \quad (\forall a, b \in G)$$

L'insieme di tutte le funzioni che soddisfano la detta definizione per i gruppi G, H si indica con $\mathrm{Hom}(G, H)$. Se la funzione f è *biiettiva*, allora si tratta di un **isomorfismo** e si indica con $G \simeq H$.

Alla luce delle precedenti definizioni possiamo affermare che un qualsiasi gruppo G contiene qualche sottogruppo isomorfo a $\mathbb{Z}$ (o a $\mathbb{Z}_n$, per qualche $n \in \mathbb{N}$).

Notazione. Nella Definizione 1.1 si è parlato di cardinalità di un insieme finito; in questo contesto si parla di ***ordine*** di un gruppo G e può essere indicato, oltre che con le notazioni già date, con $o(G)$.

Definizione 1.14 (Gruppo ciclico e generatore)**.**
Si dice **gruppo ciclico** un gruppo G in cui esiste $g \in G$ tale che $G = \langle g \rangle$, cioè per ogni $a \in G \; \exists n \in \mathbb{Z} : a = g^n$. Tale numero g è detto **generatore**.

Ricerca dei generatori. Non è nota una formula generale per calcolare i generatori dato un certo $n \in \mathbb{N}$. Tuttavia è possibile sapere quanti essi siano. Per capire come questo funzioni, sono necessarie altre due definizioni.

Definizione 1.15 (Ordine moltiplicativo di un numero)**.**
Si dice **ordine moltiplicativo** di un numero $a \in \mathbb{Z}$, dato un $n \in \mathbb{N}$, il più piccolo $k \in \mathbb{N}$ per cui vale:

$$a^k \equiv 1 \pmod{n}$$

L'ordine moltiplicativo di un numero si indica con $o_n(a)$.

Definizione 1.16 (Funzione φ di Eulero)**.**
Si definisce la funzione

$$\varphi : \mathbb{N} \to \mathbb{N}$$

$$n \mapsto \varphi(n)$$

detta **funzione φ di Eulero**, come l'applicazione che restituisce il numero degli interi coprimi con n compresi tra 1 e n. Esiste un'elegante formula per calcolare $\varphi(n)$:

$$\varphi(n) = n \cdot \prod_{p|n} \left(1 - \frac{1}{p}\right)$$

dove la notazione «$p|n$» indica che p divide n.
Il nome di tale funzione si deve al matematico svizzero Leonhard EULER (1707 – 1783).

Per calcolare quanti generatori possiede l'insieme delle classi resto $\mathbb{Z}_n$, dato un certo $n \in \mathbb{N}$ occorre sapere che, dato un numero $m \in \mathbb{N}$ candidato ad essere generatore, deve valere:

$$o_n(m) = \varphi(n) = o_n(\mathbb{Z}_n^*)$$

Se questo vale, m genera $\mathbb{Z}_n^*$. Inoltre, il numero di generatori modulo n, se esistono, è uguale a $\varphi(\varphi(n))$. È possibile fare un test per verificare se $m \pmod{n}$ genera $\mathbb{Z}_n$:

1. Calcolare $\varphi(n)$;
2. Determinare i fattori primi di $\varphi(n)$, ammettiamo siano $p_1, \ldots, p_k$;
3. Calcolare $a_i = m^{\frac{\varphi(n)}{p_i}} \pmod{n}, \quad \forall m \in \mathbb{Z}_n^*, \text{ per } i = 1, \ldots, k$;
4. Quando si trova un m per il quale tutti i valori a_i $\forall i$ risultanti dalla formula or ora enunciata siano tutti diversi da 1, allora m è un generatore.

Il seguente codice Python3 mostra una possibile esecuzione dell'algoritmo [17].

```
1 n = int(input("Inserire un numero naturale > 1 \n"))
2
3 if(n <=1):
4     print("Il numero inserito è <= 1")
5     exit()
```

```
6
7  def isPrime(n):
8      if (n == 1):
9          return False
10     for i in range(2, int((n**0.5 + 1))):
11        if(n % i == 0):
12            return False
13     else:
14             return True
15
16 def power( x, y, p):
17
18     res = 1 # Initialize result
19
20     x = x % p # Update x if it is more
21               # than or equal to p
22
23     while (y > 0):
24
25         # If y is odd, multiply x with result
26         if (y & 1):
27             res = (res * x) % p
28
29         # y must be even now
30         y = y >> 1 # y = y/2
31         x = (x * x) % p
32
33     return res
34
35 # Utility function to store prime
36 # factors of a number
37 def findPrimefactors(s, n) :
38
39     # Print the number of 2s that divide n
40     while (n % 2 == 0) :
41         s.add(2)
42         n = n // 2
43
44     # n must be odd at this po. So we can
45     # skip one element (Note i = i +2)
```

```
    for i in range(3, int(n**0.5), 2):

        # While i divides n, print i and divide n
        while (n % i == 0) :

            s.add(i)
            n = n // i

    # This condition is to handle the case
    # when n is a prime number greater than 2
    if (n > 2) :
        s.add(n)

# Function to find smallest primitive
# root of n
def findPrimitive( n) :
    s = set()

    # Check if n is prime or not
    if (isPrime(n) == False):
        return -1

    # Find value of Euler Totient function
    # of n. Since n is a prime number, the
    # value of Euler Totient function is n-1
    # as there are n-1 relatively prime numbers.
    phi = n - 1

    # Find prime factors of phi and store in a set
    findPrimefactors(s, phi)

    # Check for every number from 2 to phi
    for r in range(2, phi + 1):

        # Iterate through all prime factors of phi.
        # and check if we found a power with value 1
        flag = False
        for it in s:

            # Check if r^((phi)/primefactors)
```

```
            # mod n is 1 or not
            if (power(r, phi // it, n) == 1):

                flag = True
                break

        # If there was no power with value 1.
        if (flag == False):
            return r

    # If no primitive root found
    return -1

if(isPrime(n) == False):
    print(str(n) + " non è primo!")
else:
    if(findPrimitive(n) != -1):
        print("Il più piccolo generatore di " + str(n) + " è " +
         ↪ str(findPrimitive(n)))
```

Infine, un paio di definizioni che riguardano *anelli* e *campi*; ci saranno utili quando parleremo dell'AES.

Definizione 1.17 (Anello commutativo con identità)**.**
Un insieme R dotato delle operazioni $+$ e $\cdot$ si dice **anello commutativo con identità** se:

- $(R, +)$ è un gruppo abeliano con elemento neutro $e = 0$;
- L'operazione $\cdot$ gode delle seguenti proprietà:
  - Ha elemento neutro $e = 1 \neq 0$;
  - È associativa;
  - È commutativa;
  - Vale la proprietà distributiva, i.e.

$$(x + y) \cdot z = x \cdot z + y \cdot z, \quad \forall x, y, z \in R$$

Definizione 1.18 (Campo)**.**
Si dice **campo** un anello commutativo con identità R per cui vale che $R \setminus \{0\}$ è un gruppo rispetto alla moltiplicazione.

Si richiede quindi che ogni elemento $r \in R \,:\, r \neq 0$ abbia un inverso moltiplicativo. Questa definizione ci dovrebbe apparire in un qualche modo «naturale», in quanto sono campi $\mathbb{Q}, \mathbb{R}, \mathbb{C}$ e $\mathbb{Z}_p$ dove p è un numero primo. In quest'ultimo caso è spesso usata la notazione $\mathbb{F}_p$.

1.3 Aritmetica modulare

Vari concetti di aritmetica modulare saranno parte integrante di molti dei crittosistemi che studieremo. Mi sembra quindi opportuno spendere in questa sede qualche parola per fondare le basi di questa particolare branca della Matematica. L'aritmetica modulare fonda le sue basi sulla relazione di congruenza espressa nella Definizione 1.19. In effetti, questa relazione permette di semplificare molti calcoli – permettendo di lavorare con numeri più piccoli e quindi più gestibili – e di osservare interessanti proprietà numeriche, pensando soprattutto ai numeri primi. In effetti, la scrittura

$$a \equiv b$$

significa *semplicemente* che $a - b$ è multiplo di n. Questo ci permette di considerare ***equivalenti*** due numeri che differiscano un certo numero di volte da n. Così, $73 \equiv 15 \pmod{29}$ perché $73 - 15 = 58 = 2 \cdot 29$.

Definizione 1.19 (Classe resto)**.**
Sia dato un numero intero a. Allora la **divisione euclidea** di a per un naturale n è definita come:

$$a = k \cdot n + r \Leftrightarrow \exists!\ k, r \in \mathbb{Z}\ ;\ 0 \leq r < n \,:\, a - r = k \cdot n$$

Ciò implica che è possibile dividere l'insieme $\mathbb{Z}$ in n sottoinsiemi, dette **classi resto modulo** n. È quindi possibile stabilire la seguente *relazione di equivalenza*:

$$\alpha \equiv \beta \pmod{n} \Leftrightarrow \beta - \alpha = k \cdot n$$

La suddetta relazione è chiamata **relazione di congruenza modulo** n. Si dice infatti che α «è **congruo** a» β «modulo n».

Sono quindi definite le n **classi resto** dell'insieme $\mathbb{Z}_n := \{0, 1, \ldots, n-1\}$.

Proposizione 1 (Proprietà del calcolo modulo n)**.**
Siano

$$a \equiv \alpha \pmod{n} \quad \text{e} \quad b \equiv \beta \pmod{n} \quad \text{con } n \in \mathbb{N} \text{ e } a, b, \alpha, \beta \in \mathbb{Z}$$

(i) $a \pm b \equiv \alpha \pm \beta \pmod{n}$;

(ii) $ab \equiv \alpha\beta \pmod{n}$.

Dimostrazione.

(i) Siccome $a \equiv \alpha \pmod{n}$, per definizione significa che $a - \alpha = kn \quad (k \in \mathbb{Z})$; lo stesso vale per b: $b \equiv \beta \pmod{n} \Leftrightarrow b - \beta = ln \quad (l \in \mathbb{Z})$. Pertanto si ha che:

- $a + b = (kn + \alpha) + (ln + \beta)$ e
- $\alpha + \beta = (a - kn) + (b - ln)$

Riordinando si ottiene $(a + b) - (\alpha + \beta) = 2n(k + l)$, cioè $(a + b) - (\alpha + \beta)$ è multiplo di n, come richiesto. Per la dimostrazione con il «$-$» basta ragionare analogamente con gli opposti. ✓

(ii) Partiamo sempre dalla definizione: $ab - \alpha\beta$ dev'essere un multiplo di n. Calcoliamo ora $ab - \alpha\beta$ sapendo che $a = kn + \alpha$ e $b = ln + \beta$. Otteniamo:

$$(kn + \alpha)(ln + \beta) - \alpha\beta = kln^2 + kn\beta + ln\alpha + \alpha\beta - \alpha\beta = (kln + k\beta + l\alpha)n$$

cioè $ab - \alpha\beta$ è un multiplo di n, come richiesto. ✓ ■

È importante osservare che le congruenze possono anche contenere delle incognite e quindi essere risolte come delle equazioni. Il lettore deve però prestare attenzione al fatto che, *in generale*, non è possibile sfruttare la proprietà di cancellazione, a meno che i due membri della congruenza non siano **coprimi**[1] con il modulo. Si prenda ad esempio la congruenza

$$42 \equiv 12 \pmod{10}$$

La congruenza enunciata è vera, siccome $42 - 12 = 30 = 10 \cdot 3$. Il fattore 6 è comune ad entrambi i membri della congruenza, tuttavia non è possibile cancellarlo. Non è vero, infatti, che: $7 \equiv 2 \pmod{10}$.

Questa osservazione si giustifica mediante la seguente

Proposizione 2 (Legge di cancellazione)**.**
In un gruppo abeliano $(G, \diamond)$, presi $a, b, c \in G$, vale la **legge di cancellazione**, cioè

$$a \diamond b = a \diamond c \Rightarrow b = c$$

se e solo se $a \in G^*$, cioè a è un elemento invertibile

Dimostrazione. Vale: $a \diamond b = a \diamond c \Leftrightarrow (a^{-1} \diamond a) \diamond b = (a^{-1} \diamond a) \diamond c \Leftrightarrow e \diamond b = e \diamond c$ e quindi $b = c$. ■

[1] Due numeri $a, b \in \mathbb{N}$ si dicono **coprimi** (o *primi fra loro*) se vale $\text{MCD}(a, b) = 1$, spesso semplicemente indicato con $(a, b) = 1$.

La legge di cancellazione è una generalizzazione del concetto di *invertibilità*. Ciò significa che la legge può valere solo ove esiste l'inverso di ogni elemento rispetto all'operazione che si vuole applicare.

In effetti vale la:

Proposizione 3.
Sia $a \in \mathbb{Z}_n$. Allora, a è invertibile $\Leftrightarrow \text{MCD}(a, n) = 1$.

Dimostrazione. Siano $\alpha, \beta \in \mathbb{Z}$. Allora vale:

$$\alpha a + \beta n = 1 \Leftrightarrow \alpha a = 1 - \beta n \Leftrightarrow \alpha a \equiv 1 \pmod{n} \Leftrightarrow \exists \alpha \in \mathbb{Z}_n \text{ che è l'inverso di } a \quad \blacksquare$$

Osservazione.

$$\mathbb{Z}_n^* = \mathbb{Z}_n \setminus \{0\} \Leftrightarrow n \text{ è primo}$$

dove il simbolo «*» indica che sono stati rimossi gli elementi che non hanno inverso. Dal momento che tutti gli elementi di $\mathbb{Z}_n$ tranne lo 0 hanno un inverso, allora $\mathbb{Z}_n^*$ è un **campo** e quindi n è primo.

Torniamo ora a parlare della possibilità di considerare, fatte le dovute proporzioni, delle congruenze come delle equazioni. Ci occuperemo del caso più semplice, ossia delle congruenze *lineari*, paragonabili quindi a delle equazioni di primo grado. La più generica fra queste è ovviamente la

$$ax \equiv b \pmod{n}$$

dove $a, b \in \mathbb{Z}$ e $n \in \mathbb{N}$ sono noti e x è l'incognita. Sia, per esempio, la congruenza lineare

$$5x \equiv 7 \pmod{12}$$

Dal momento che non ci troviamo di fronte ad una normale equazione in $\mathbb{R}$, ma a una congruenza lineare in $\mathbb{Z}_{12}$, è necessario dare a x i valori contenuti nell'insieme delle classi resto modulo 12. Siccome $\mathbb{Z}_{12} = \{0, 1, \ldots, 11\}$, il termine $5x$ assume valori:

$$0, 5, 10, \ldots, 55$$

Naturalmente questi ultimi vanno ridotti modulo 12, per ottenere:

$$0, 5, 10, 3, 8, 1, 6, 11, 4, 9, 2, 7$$

Il valore 7 è ottenuto per $x = 11$, quindi $x = 11$ è **una** soluzione della congruenza; sono soluzioni anche **tutti gli altri valori congrui a 11 modulo 12**.

1.4 Definizione di *sicurezza*

In questo testo diremo più volte che un certo algoritmo crittografico è «*sicuro*» o «*più sicuro* rispetto ad un altro», ecc. ***Ma che cos'è veramente la sicurezza?*** Non esiste una definizione univoca del concetto di sicurezza. Inoltre, la sicurezza totale – cioè «che non presenta rischi» – in realtà non esiste. La sicurezza è quindi un **concetto relativo** che va rapportato al contesto in cui se ne parla. Il grado di sicurezza è infatti variabile, a seconda dell'importanza delle informazioni che si desiderano proteggere.

Viene quindi da sé che i processi da attuare per proteggere le informazioni varino al variare del grado di sicurezza. Questa considerazione ci porta al solito compromesso *sicurezza vs. usabilità*, fil rouge del mio Lavoro e, spesso, dell'evoluzione dei crittosistemi. Il *costo* – economico o computazionale che sia – generato dalla necessità di proteggere delle informazioni dev'essere commisurato all'importanza delle informazioni stesse, dal danno che si potrebbe causare se queste dovessero essere rese pubbliche e dalla facilità con cui si vuole accedere alle stesse. In effetti, accessi più sicuri possono voler dire rinunciare a qualche comodità e dover utilizzare, per esempio, un'autenticazione multifattoriale (es. password + OTP[2]).

Definizione 1.20 (Sicurezza perfetta (di Shannon))**.**
Si dice sicurezza **perfetta** (o sicurezza **di Shannon**) la sicurezza offerta da un algoritmo crittografico che rende il messaggio in chiaro m e il messaggio cifrato c *statisticamente indipendenti*, ossia:

1. Non si devono poter ottenere informazioni su m analizzando c;
2. L'incertezza che si ha su m dopo aver osservato c dev'essere la stessa che si aveva prima di aver osservato c.

È importante notare innanzitutto che, in un cifrario perfetto, la lunghezza della chiave k scelta dev'essere maggiore o uguale a quella del messaggio m da cifrare. Il nome si deve all'ingegnere e matematico statunitense Claude E. SHANNON (1916 – 2001). Per parlare in maniera più approfondita della sicurezza di Shannon ed enunciare in maniera più formale i suoi risultati, è necessaria una conoscenza di base di alcuni concetti di probabilità. Nella sezione seguente richiamerò quelli più importanti.

1.5 Richiami di probabilità

La **probabilità** è una disciplina della Matematica che si occupa di studiare gli ***esiti*** di un particolare *esperimento aleatorio*; classifica gli esiti che hanno caratteristiche comuni in

[2]OTP: *One-Time Password*, lett. «password monouso», Un codice, di solito un numero di 4 o 6 cifre, generato casualmente e valido per un solo accesso. Normalmente è inviato via SMS o tramite un'applicazione per *smartphone*.

eventi e studia la frequenza con cui essi accadono. Spesso si è infatti interessati a questo tipo di informazioni, piuttosto che all'esito di un particolare evento, soprattutto quando il numero di questi ultimi è elevato.

Definizione 1.21 (Spazio campionario)**.**
Si dice **spazio campionario** (o *spazio campione*) l'insieme che contiene tutti gli esiti di un esperimento dato. Lo spazio campionario si indica con il simbolo Ω.

Definizione 1.22 (Classica di probabilità)**.**
La probabilità $\mathbb{P}$ di un evento E è definita *classicamente* come il **rapporto tra gli esiti «favorevoli»**, cioè in cui E si verifica, **e gli esiti «possibili»**, cioè $\text{card}(\Omega)$. In simboli:

$$\mathbb{P}(E) = \frac{n_E}{\text{card}(\Omega)}$$

Questa definizione è problematica, perché non è una definizione propriamente detta, in quanto presenta un elemento di circolarità: essa vale infatti soltanto nel caso particolare in cui tutti gli esiti siano *equiprobabili*, cioè abbiano **uguale probabilità**. Ecco la circolarità: si usa un concetto di probabilità per definire cos'è la probabilità.

La definizione precedente è stata chiamata «classica» perché è la prima, e la più semplice, che è stata data. Modernamente si utilizzano i seguenti concetti:

- Variabile aleatoria;
- Distribuzione di probabilità.

Definizione 1.23 (Variabile aleatoria)**.**
Si dice **variabile aleatoria** una funzione

$$\mathbf{X} : \Omega \to \mathbb{R}$$

Se Ω è finito o infinito numerabile, $\mathbf{X}$ si dice variabile aleatoria *discreta*, altrimenti si usa il termine *continua*.

Definizione 1.24 (Distribuzione di probabilità)**.**
Si dice **distribuzione di probabilità** di una variabile aleatoria discreta la funzione

$$f(a) = \mathbb{P}(\mathbf{X} = a) \quad (a \in \mathbb{R})$$

Per discutere la teoria della sicurezza di Shannon è fondamentale conoscere le definizioni di **probabilità composta** e **probabilità condizionata**. Enunciamole:

Definizione 1.25 (Probabilità composta [5])**.**
Siano date due variabili aleatorie $\mathbf{X}$ e $\mathbf{Y}$ definite sugli insiemi finiti X e Y, con $x \in X$ e $y \in Y$. La **probabilità composta** $\mathbb{P}(x, y) = \mathbb{P}(\mathbf{X} = x, \mathbf{Y} = y)$ è la probabilità che $\mathbf{X}$ assuma il valore x **e** $\mathbf{Y}$ assuma il valore y.

Nota: **X** e **Y** sono dette *indipendenti* se vale $\mathbb{P}(x, y) = \mathbb{P}(x)\mathbb{P}(y) \quad (\forall x \in X, y \in Y)$.

Definizione 1.26 (Probabilità condizionata [5])**.**
Siano date due variabili aleatorie **X** e **Y** definite sugli insiemi finiti X e Y, con $x \in X$ e $y \in Y$. La **probabilità condizionata** $\mathbb{P}(x|y)$ (si legga: «*x dato y*») è la probabilità che **X** assuma il valore x dato il fatto che **Y** assuma il valore y. Vale la seguente proprietà che lega proprietà composta e condizionata: $\mathbb{P}(x, y) = \mathbb{P}(x|y) \cdot \mathbb{P}(y)$.

I due concetti sopra descritti sono correlati anche dall'importante

Teorema 1 (Bayes)**.**
Se $\mathbb{P}(y) \neq 0$, allora vale:

$$\mathbb{P}(x|y) = \frac{\mathbb{P}(x)}{\mathbb{P}(y)} \cdot \mathbb{P}(y|x)$$

Il nome si deve al matematico britannico Thomas BAYES (1702 – 1761).

1.6 Sicurezza di Shannon

Torniamo a discutere la *sicurezza di Shannon*, la cui definizione è data nella 1.20.

È possibile descrivere la sicurezza di un crittosistema in tre modi distinti [5]:

1. **Sicurezza computazionale** Un crittosistema è detto *computazionalmente sicuro* se il miglior algoritmo che permette di violarlo ha complessità computazionale superiore ad un certo limite N fissato, sufficientemente grande. Allo stato attuale, nessun algoritmo crittografico utilizzato è stato dimostrato essere computazionalmente sicuro;

2. **Sicurezza dimostrabile** Un crittosistema è detto *dimostrabilmente sicuro* se è possibile fornire una prova del fatto che la sua sicurezza sia equivalente a quella di un problema ritenuto «difficile da risolvere». Si parla quindi di *sicurezza relativa* (a quella di un altro problema);

3. **Sicurezza incondizionale** Un crittosistema è detto *sicuro incondizionalmente* se non è violabile, data una potenza di calcolo illimitata.

Shannon si è focalizzato sullo studio di quest'ultimo tipo di crittosistemi e lo ha fatto sfruttando la teoria della probabilità. Il lavoro completo da egli svolto è l'opera [6].

Sia dato l'insieme dei messaggi in chiaro $\mathfrak{M}$ e su di esso sia definita la variabile aleatoria **M**. La probabilità che un certo messaggio in chiaro m sia scelto è quindi $\mathbb{P}(\mathbf{M} = m)$. Facciamo lo stesso anche per l'insieme delle chiavi $\mathfrak{K}$ e la variabile aleatoria **K**. Sup-

poniamo che le due variabili aleatorie siano indipendenti, ossia la probabilità di una non influenza quella dell'altra, dal momento che la chiave è scelta prima del messaggio da cifrare. In $\mathfrak{C}$ viene indotta una nuova distribuzione di probabilità: se $c = f_k(m)$ è il messaggio cifrato, si ha che:

$$\mathbb{P}(\mathbf{C} = c) = \sum_{k\,:\,c \in C(k)} \mathbb{P}(\mathbf{K} = k) \cdot \mathbb{P}(\mathbf{M} = f_k^{-1}(c))$$

L'insieme $C(k)$ è l'insieme di tutti le immagini di f_k. La probabilità che sia c il testo cifrato «scelto» dipende sia dalla probabilità che sia scelta una chiave k, sia dalla probabilità che $c \in C(k)$. Notiamo che è stata applicata la definizione di probabilità composta: siccome $\mathbf{K}$ e $\mathbf{M}$ sono indipendenti, allora vale $\mathbb{P}(k, m) = \mathbb{P}(k) \cdot \mathbb{P}(m)$. Si parla, quindi, della probabilità che k sia la chiave scelta **e** che m sia il testo in chiaro.

Conseguentemente, la probabilità che c sia il testo cifrato di m dato è calcolata dalla probabilità condizionata:

$$\mathbb{P}(\mathbf{C} = c | \mathbf{M} = m) = \sum_{k\,:\,m = f_k^{-1}(c)} \mathbb{P}(\mathbf{K} = k)$$

Possiamo ora applicare il Teorema di Bayes per calcolare $\mathbb{P}(\mathbf{M} = m | \mathbf{C} = c)$. Infatti si ha:

$$\mathbb{P}(\mathbf{M} = m | \mathbf{C} = c) = \frac{\mathbb{P}(\mathbf{M} = m) \cdot \sum\limits_{k\,:\,m = f^{-1}(c)} \mathbb{P}(\mathbf{K} = k)}{\sum\limits_{k\,:\,c \in C(k)} \left(\mathbb{P}(\mathbf{K} = k) \cdot \mathbb{P}(\mathbf{M} = {f_k}^{-1}(c))\right)}$$

Possiamo ora riprendere la definizione 1.20 e renderla più formale.

Definizione 1.27 (Crittosistema di Shannon)**.**
Un crittosistema si dice **perfetto** (o *di Shannon*) se e solo se

$$\mathbb{P}(m|c) = \mathbb{P}(m) \quad \forall m \in \mathfrak{M}, c \in \mathfrak{C} \iff \mathbb{P}(c|m) = \mathbb{P}(c)$$

Tradotta in italiano, la precedente definizione si leggerebbe così: «la probabilità che un certo m appartenente a $\mathfrak{M}$ sia scelto (fra gli altri) dato il fatto che è stato scelto un c appartenente a $\mathfrak{C}$ è uguale alla probabilità che un certo m sia scelto da $\mathfrak{M}$ senza nulla sapere di c. Cioè, dalla scelta di c **non si può evincere nulla sulla scelta di** m». È chiaro che vale anche il contrario.

Queste considerazioni ci portano all'enunciazione del

Teorema 2 (Shannon)**.**
Sia $(\mathfrak{M}, \mathfrak{C}, f, f^{-1})$ un crittosistema e sia $\mathfrak{K}$ l'insieme finito delle chiavi. Si abbia che $\text{card}(\mathfrak{M}) = \text{card}(\mathfrak{C}) = \text{card}(\mathfrak{K})$ e che $\mathbb{P}(m) > 0, \quad \forall m \in \mathfrak{M}$. Allora tale crittosistema è perfetto se e solo se ogni chiave è usata con una probabilità equa $\frac{1}{\text{card}(\mathfrak{K})}$ e $\forall m \in \mathfrak{M}, c \in \mathfrak{C}$ esiste un'unica chiave $k \in \mathfrak{K}$ tale che $f_k(m) = c$.

Per la dimostrazione si veda [5], p. 45.

2 Storia della crittografia dagli albori al XX secolo

2.1 Introduzione

La Crittografia è una disciplina la cui storia segue quella dell'uomo dalla notte dei tempi. La necessità di proteggere le informazioni ci accompagna da sempre e, nonostante si sia modificata nelle modalità, non ha mai perso la sua posizione importante. Chiaramente, nel XXI secolo la crittografia è fondamentale come mai prima d'ora, poiché proteggere il sempre maggiore numero di informazioni scambiate con i vari mezzi di comunicazione, Internet su tutti, necessita di tecniche sempre più avanzate.

Fino all'avvento dell'alfabetizzazione generale la protezione delle informazioni era facilitata dal fatto che poche persone sapevano leggere e scrivere. Ad oggi, la crittografia deve servirsi di metodi matematici avanzati. Di questo ci occuperemo nel prossimo capitolo; qui parliamo delle tecniche crittografiche che si sono applicate dal V secolo a.C. al 2000.

Le prime testimonianze di metodi crittografici si hanno nelle *Storie* di ERODOTO (484 a.C. – 430 a.C.). In questo caso si parla di una guerra tra Spartani e Persiani, in cui un testo scritto su una tavoletta di cera fu ricoperto da un ulteriore strato dello stesso materiale per farla sembrare vuota, vergine. Secondo l'accordo tra mittente e destinatario, quest'ultimo aveva il compito di grattare lo strato supplementare di cera e leggere il messaggio. Seppure questo è considerato il primo esempio di crittografia, in realtà si tratta di steganografia, dal greco στεγανός (coperto) e γραφία (scrittura), cioè «*nascondere la scrittura*». La differenza fondamentale tra queste due discipline è che lo scopo della steganografia è quello di occultare l'informazione «coprendola» con un'altra che ha un significato a sé stante; d'altra parte la crittografia nasconde le informazioni con sequenze di caratteri che non possiedono senso compiuto. In questo contesto si inseriscono pure gli Antichi Egizi che avevano un alfabeto di geroglifici crittografati, cioè diversi da quelli comunemente in uso.

Si può dire che la steganografia è l'antenata della crittografia, ma poi queste due discipline hanno avuto sviluppi separati e sono entrambe attuali. La crittografia infatti serve ad occultare il contenuto di un messaggio; la steganografia nasconde invece il fatto che il messaggio sia stato inviato. È possibile per esempio nascondere un file di testo (`.txt`) in un file immagine (`.jpeg`).

2.2 Crittografia monoalfabetica e polialfabetica

Prima di addentrarci nello studio dei vari metodi crittografici che hanno caratterizzato l'evoluzione della risposta al bisogno di protezione delle informazioni, è utile e doveroso discriminare fra crittografia **monoalfabetica** e crittografia **polialfabetica**. Conosceremo

la differenza tra crittografia simmetrica e asimmetrica nel capitolo 3; ora apprendiamo che nella crittografia simmetrica esistono due modi fondamentali di agire. Il primo, caratterizzante della crittografia monoalfabetica, consiste nel sostituire ogni occorrenza di $\alpha \in m$ con lo stesso simbolo in c in funzione della chiave k scelta. Questo rende ogni cifrario monoalfabetico suscettibile all'analisi di frequenza (per un approfondimento si veda il sottocapitolo sul Cifrario di Cesare). Per ovviare a questa importante lacuna di sicurezza si è ricorsi alla crittografia polialfabetica. Questo metodo consiste nel sostituire un simbolo $\alpha \in m$ con un simbolo $\beta \in c$ che cambia secondo la chiave k scelta. Uno dei più celebri esempi di crittografia polialfabetica è il Cifrario di Vigenère.

2.3 Il Cifrario di Cesare

Il Cifrario di Cesare (100 a.C – 44 a.C.) è un metodo molto semplice per crittografare un messaggio. Fu inventato da Giulio Cesare. Si prende ogni carattere del messaggio in chiaro m e lo si sostituisce con la lettera che sta tre posti più avanti nell'alfabeto $\mathfrak{A}$ prescelto. Così, dato il messaggio $m = \texttt{casa}$ si ottiene il messaggio cifrato $c = \texttt{FDVD}$ secondo l'alfabeto italiano di 21 lettere. Variazioni del metodo di Cesare possono essere usate scegliendo una chiave $k \neq 3$ (con $1 < k < \text{card}(\mathfrak{A})$). La scelta $k = 0$ è sconsigliata perché è triviale ($m = c$), così come le scelte $k = 1$ (a ogni lettera corrisponde la successiva) e $k = \text{card}(\mathfrak{A})$ perché $x \equiv x \pmod{k}\ \forall x$.

Ho realizzato un piccolo programma console Java che cripta il messaggio con il cifrario di Cesare, secondo una chiave k fornita dall'utente ($1 \leq k \leq 26$). Di seguito il listato:

```java
//package lam; Uncomment this line and change the package name according to your settings. If you are executing this online, leave this line commented.

import java.util.Scanner;

public class MyClass {

    public static void main(String[] args) {

        Scanner input = new Scanner(System.in);

        String plaintext = "";
        String criptato = "";
        int k = 0; // chiave
        System.out.println("Inserire una chiave [numero intero fra 1 e 26]");

```

```
16          try {
17
18              k = input.nextInt();
19
20
21          } catch (Exception e) {
22              System.out.println("La chiave inserita non è un numero intero.
                ↪  Riprovare. PROGRAMMA TERMINATO.");
23              System.exit(-1);
24          } finally {}
25
26          if (k < 0 || k > 26) {
27              System.out.println("La chiave è minore di 0 o maggiore di 26.
                ↪  Riprovare. PROGRAMMA TERMINATO.");
28              System.exit(-1);
29          } else {
30              System.out.println("Inserire un messaggio da criptare (gli spazi
                ↪  non contano)");
31              plaintext = input.next().toUpperCase(); // uso solo lettere
                ↪  maiuscole per semplificare.
32
33
34              int[] plainarray = new int[plaintext.length()];
35              int[] arraycriptato = new int[plaintext.length()];
36
37              for (int i = 0; i < plaintext.length(); i++) {
38                  plainarray[i] = (int) plaintext.charAt(i);
39              }
40
41              for (int i = 0; i < plaintext.length(); i++) {
42
43                  arraycriptato[i] = (plainarray[i] + k);
44                  if (arraycriptato[i] > 90) {
45                      arraycriptato[i] -= 26;
46                  }
47                  criptato += (char) arraycriptato[i];
48
49              }
50              System.out.println("MESSAGGIO CRIPTATO: " + criptato + "\t [chiave
                ↪  " + k + "]");
```

```
51              input.close();
52          }
53
54      }
55
56  }
```

Questo metodo crittografico è considerato poco sicuro. All'epoca della sua creazione poteva essere considerato affidabile perché a saper leggere e scrivere era una piccolissima parte della popolazione. Oggi non è più così, ed è evidente come questo metodo sia suscettibile all'*analisi di frequenza*. Il problema maggiore che non riguarda solo questo metodo, ma tutti quelli che fanno uso della *cifratura monoalfabetica*, è che ogni glifo di m ha uno e un solo corrispondente in c. Ciò significa che, soprattutto se conosciamo la lingua in cui m è scritto e m è sufficientemente lungo, possiamo cercare di ottenere m sapendo che alcune lettere appaiono, in una certa lingua, più frequentemente di altre. Per l'italiano vale che le lettere «E», «A», «I» e «O» sono le più frequenti [13].

È quindi facile iniziare «per tentativi» ad associare le lettere che appaiono più di frequente in c con le vocali. Non è una garanzia di successo, ma è un buon punto di partenza. Sapendo poi che la chiave è un numero fra 1 e 26, è possibile tentare un attacco di *forza bruta* (*brute force* in inglese) e tentare tutte le possibili k. Mettendo in conto che questo lavoro può al giorno d'oggi essere lasciato a un semplice PC dell'elettronica di consumo, data la complessità computazionale polinomiale, la cattiva sicurezza di questo metodo mi sembra provata.

2.4 Il Cifrario di Vigenère

Se con il Cifrario di Cesare abbiamo trattato un'applicazione della crittografia monoalfabetica, con il Cifrario di Vigenère poniamo ora il nostro sguardo su un metodo che implica l'utilizzo della più robusta cifratura polialfabetica. Questo cifrario è stato inventato dal diplomatico, crittografo, traduttore e alchimista francese Blaise DE VIGENÈRE (1523 – 1596). Si tratta del più semplice dei cifrari polialfabetici.

Decidiamo innanzitutto con quale alfabeto lavorare; per questo esempio useremo l'alfabeto inglese di 26 lettere.
Dobbiamo poi scegliere una chiave (spesso detta *verme*). Se la lunghezza della chiave è minore rispetto a quella del testo in chiaro, come accade il più delle volte, essa viene replicata (e infine troncata se necessario) per ottenere la stessa lunghezza del messaggio. Quindi se $m =$ «`messaggio`» e $k =$ «`CHIAVE`», allora come verme si utilizzerà

«CHIAVECHI». Costruiamo poi la seguente matrice 27×27 dove, per ogni riga, l'alfabeto A-Z è traslato di una posizione. Ovviamente, la matrice ha ordine $\text{card}(\mathfrak{A}) + 1$.

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

Tabella 1: Tabella del Cifrario di Vigenère.

Il metodo di cifratura è semplicemente $c = a + b \pmod N$ dove a, b sono i numeri che corrispondono alle lettere rispettivamente del testo in chiaro e della chiave, c è il numero della lettera nel testo cifrato e $N = \text{card}(\mathfrak{A})$. Per decifrare vale ovviamente $a = n - b + N$. Proviamo quindi a crittografare $m =$ «crittografia» con la chiave k. Otteniamo $c =$ «EYQTOSIYIFDE».

Ho realizzato un programma Java che cripta un messaggio a scelta con una chiave a scelta usando l'alfabeto di 26 lettere.

```java
//package lam; Uncomment this line and change the package name according to
 ↪  your settings. If you are executing this online, leave this line
 ↪  commented.

import java.util.Scanner;

public class MyClass {

    public static void main(String[] args) {


        //Cifrario di Vigenère secondo l'alfabeto inglese di 26 lettere.

        String chiave = "";
        String plaintext = "";
        Scanner input = new Scanner(System.in);



        System.out.println("CIFRARIO DI VIGENÈRE SECONDO L'ALFABETO INGLESE");
        System.out.println("\n");
        System.out.println("Inserire il testo da cifrare.");

        plaintext = input.next().toUpperCase(); // Uso le maiuscole per
         ↪  semplicità

        System.out.println("Inserire una chiave");

        chiave = input.next().toUpperCase(); // Uso le maiuscole per
         ↪  semplicità

        if (chiave.length() < plaintext.length()) {
            int differenza = plaintext.length() - chiave.length();

            for (int i = 0; i < differenza; i++) {

                chiave += chiave.charAt(i);
            }


```

```
37
38              System.out.println("Siccome la lunghezza della chiave è minore di
                ↪  quella del testo in chiaro, la chiave viene iterata per
                ↪  ottenere: " + chiave);
39
40          }
41
42          input.close();
43
44          int[] chiave_array = new int[chiave.length()];
45          int[] plain_array = new int[plaintext.length()];
46
47          // System.out.println(plaintext + "\t" + chiave);
48
49          for (int i = 0; i < chiave.length(); i++) {
50              chiave_array[i] = (int) chiave.charAt(i);
51          }
52
53          for (int i = 0; i < plaintext.length(); i++) {
54              plain_array[i] = (int) plaintext.charAt(i);
55          }
56
57          int[] cifrato_array = new int[plaintext.length()];
58
59          String cifrato = "";
60
61          for (int i = 0; i < plaintext.length(); i++) {
62
63              cifrato_array[i] = (plain_array[i] + chiave_array[i]) % 26 + 65;
                ↪  // Lavoro mod 26 (alfabeto inglese) e aggiungo 65 perché il
                ↪  codice ASCII della "A" maiuscola è 65.
64
65              cifrato += (char) cifrato_array[i];
66          }
67
68          System.out.println("\n \n TESTO CIFRATO: " + cifrato);
69
70      }
71
72  }
```

2.5 Il Metodo di Hill

La cifratura secondo il Metodo di Hill è un tipo di cifratura simmetrica basata su concetti elementari di algebra lineare. Questo metodo crittografico è stato inventato dal matematico americano Lester HILL (1891 – 1961).
Ammettiamo di voler criptare il messaggio $m =$ «`crittografia`». Come per ogni metodo crittografico, ci servirà una chiave. La chiave sarà una matrice quadrata (cioè dell'ordine $j \times j, j \geq 2$). Gli elementi della matrice chiave K dovranno appartenere a un certo $\mathbb{Z}_n$, dove $n = \text{card}(\mathfrak{A})$. Se scegliamo l'alfabeto inglese, allora $n = 26$. In realtà, sarebbe auspicabile scegliere un n primo perché così facendo si scongiura il rischio di avere fattori comuni con il modulo nel determinante di K. Questo impedirebbe infatti la decifratura; per decifrare si usa infatti la matrice inversa K^{-1} di K. Per invertire una matrice è necessario che:

- il suo determinante sia non nullo, se gli elementi sono reali o complessi;
- il suo determinante deve essere coprimo con n per lo $\mathbb{Z}_n$ scelto.

Per questo si preferisce usare delle matrici per cui $\det(K) = \pm 1$ perché ± 1 sono invertibili in ogni $\mathbb{Z}_n$.

L'ordine m della matrice è a scelta del mittente. Per semplicità, nei prossimi esempi useremo $m = 2$.

La funzione f di cifratura del Metodo di Hill è la seguente:

$$f\begin{pmatrix} x \\ y \end{pmatrix} = K \cdot \begin{pmatrix} x \\ y \end{pmatrix}$$

La funzione di decifratura è ovviamente

$$f^{-1}\begin{pmatrix} x \\ y \end{pmatrix} = K^{-1} \cdot \begin{pmatrix} x \\ y \end{pmatrix}$$

È chiaro che l'ordine della matrice argomento è lo stesso di quello della matrice K. Questa considerazione deriva direttamente dalle proprietà della moltiplicazione matriciale. Infatti, si possono moltiplicare due matrici se il numero di colonne della prima è pari a quello di righe della seconda. Si ricorda inoltre che la moltiplicazione tra matrici non è commutativa.

Riprendiamo il nostro esempio con $m =$ «`crittografia`»; scegliamo una matrice K con $\det(K) = 1$, per esempio:

$$K = \begin{pmatrix} 5 & 23 \\ -2 & -9 \end{pmatrix}$$

Abbiamo, implicitamente, scelto che il messaggio andrà diviso in blocchi di due caratteri, visto che l'ordine di K è 2. Se il numero di caratteri di m fosse dispari o, in generale, non divisibile per l'ordine della matrice, si possono aggiungere dei caratteri nulli che poi andranno troncati dal messaggio cifrato. In un programma si potrebbe concatenare alla stringa m il numero necessario di caratteri `NUL` (ASCII 0).

Dividiamo m in sottostringhe di 2 caratteri: `cr|it|to|gr|af|ia`. Costruiamo ora i vettori colonna $\begin{pmatrix} x \\ y \end{pmatrix}$ inserendo il numero corrispondente alla lettera nell'alfabeto scelto (in questo caso l'alfabeto inglese di 26 lettere). Avremo quindi:

$$\texttt{cr} \to \begin{pmatrix} 3 \\ 18 \end{pmatrix} \texttt{it} \to \begin{pmatrix} 9 \\ 20 \end{pmatrix} \texttt{to} \to \begin{pmatrix} 20 \\ 15 \end{pmatrix} \texttt{gr} \to \begin{pmatrix} 7 \\ 18 \end{pmatrix} \texttt{af} \to \begin{pmatrix} 1 \\ 6 \end{pmatrix} \texttt{ia} \to \begin{pmatrix} 9 \\ 1 \end{pmatrix}$$

Calcolando l'immagine di ciascuno dei sei argomenti e concatenando ognuno dei pacchetti immagine in una stringa, otteniamo $c =$ «`MLKPCSGTMDPA`».

Lo stesso risultato è ottenibile con il seguente programma Java da me scritto:

```java
//package lam; Uncomment this line and change the package name according to your settings. If you are executing this online, leave this line commented.

import java.util.Scanner;

public class MyClass {

    public static void main(String[] args) {

        Scanner input = new Scanner(System.in);

        int[][] cifrante = new int[2][2];
        System.out.println("Questo programma cripta una parola con il metodo di Hill data una matrice di ordine 2. \n \n ");
        System.out.println("Inserire nell'ordine (riga per riga) i 4 coefficienti della matrice cifrante");
        int a = input.nextInt();
        int b = input.nextInt();
        int c = input.nextInt();
        int d = input.nextInt();


```

```
20
21          if (a * d - b * c != 1 && a * d - b * c != -1) {
22              System.out.println("La matrice ha determinante diverso da +/- 1 e
                ↪  pertanto non è valida. PROGRAMMA TERMINATO");
23              System.exit(0);
24          }
25          cifrante[0][0] = a;
26          cifrante[0][1] = b;
27          cifrante[1][0] = c;
28          cifrante[1][1] = d;
29
30
31
32          System.out.println("Inserire la parola da crittografare.");
33          String messaggio = input.next().toUpperCase();
34
35
36          if (messaggio.length() % 2 != 0) {
37              System.out.println("La stringa ha un numero dispari di caratteri.
                ↪  PROGRAMMA TERMINATO");
38          }
39
40          input.close();
41
42          String[] splitmessaggio;
43          splitmessaggio = messaggio.split("(?<=\\G..)", -1); //Con questa regex
             ↪  divido il messaggio in stringhe di 2 caratteri.
44
45          int x = 0, y = 0;
46
47          String cifrato = "";
48
49          for (int i = 0; i < splitmessaggio.length - 1; i++) {
50              try {
51                  x = (int) splitmessaggio[i].charAt(0) - 64;
52                  y = (int) splitmessaggio[i].charAt(1) - 64;
53
54              } catch (Exception ignore) {}
55
56
```

```
57              // System.out.println("x = " + x + " y = " + y);
58
59              int[][] f = new int[2][1];
60
61              f[0][0] = Math.abs((cifrante[0][0] * x + cifrante[0][1] * y)) %
                ↪  26;
62              f[1][0] = Math.abs((cifrante[1][0] * x + cifrante[1][1] * y)) %
                ↪  26;
63
64
65              cifrato += (char)(f[0][0] + 64);
66              cifrato += (char)(f[1][0] + 64);
67
68          }
69
70          System.out.println(" \n  \n MESSAGGIO CIFRATO: " + cifrato);
71      }
72
73  }
```

2.6 Il Cifrario di Vernam

Il Cifrario di Vernam, inventato dall'ingegnere americano Gilbert Sanford VERNAM nel 1917, è una variazione del Cifrario di Vigenère in cui il requisito aggiuntivo consiste nel fatto che la chiave dev'essere lunga esattamente quanto il testo in chiaro e dev'essere usata una sola volta (da qui il nome inglese «***One-Time** pad*»). Vien detto sistema crittografico *perfetto* dove questo aggettivo risponde alla definizione data dall'ingegnere e matematico statunitense Claude Edwood SHANNON, secondo cui un cifrario è detto *perfetto* se, una volta crittografato, del messaggio in chiaro è ottenibile solo la sua lunghezza.

Questo cifrario, considerando come caso particolare quello in cui è usato l'alfabeto inglese di 26 lettere, è un esempio di cifrario perfetto (secondo Shannon) perché il crittosistema è

$$(\mathfrak{A}^{26}, \mathfrak{A}^{26}, f, f^{-1}) \quad \text{e} \quad \mathfrak{K} = \mathfrak{A}^{26}.$$

È ovvio che le cardinalità siano le stesse per tutti gli insiemi (si tratta dello stesso insieme $\mathfrak{A}^{26}$ e che ogni chiave $k \in \mathfrak{K}$ ha probabilità d'esser scelta $\frac{1}{\text{card}(\mathfrak{K})}$, poiché ogni elemento di $\mathfrak{K}$ è indipendente dagli altri. Si può pensare come un gioco: scegliere una parola a caso, data una lunghezza l. Ogni parola di lunghezza l ha la stessa probabilità di essere

scelta, così come le n facce di un dado equo hanno la stessa probabilità $\frac{1}{n}$ di presentarsi al giocatore. Inoltre, una sola chiave permette la corretta decrittazione.

Il funzionamento è in tutto e per tutto uguale a quello del Cifrario di Vigenère (si veda il sottocapitolo precedente).

Di seguito il codice Java del Cifrario di Vernam.

```java
//package lam; Uncomment this line and change the package name according to your settings. If you are executing this online, leave this line commented.

import java.util.Scanner;

public class MyClass {

    public static void main(String[] args) {

        System.out.print("Questo programma cifra un messaggio con il cifrario di Vernam data una chiave. \n \n");

        Scanner input = new Scanner(System.in);

        System.out.println("Inserire la parola da cifrare.");

        String messaggio = input.next().toUpperCase(); // Uso solo le maiuscole per semplicita'.

        System.out.println("Inserire la chiave.");

        String chiave = input.next();

        if (chiave.length() < messaggio.length()) {
            System.out.println("Chiave non valida. La lunghezza della chiave e' minore di quella del messaggio.");
            System.exit(0);
        } else {
            chiave = chiave.toUpperCase(); // Uso solo le maiuscole per semplicita'.
        }
        input.close();

```

```
29          int[] intmessaggio = new int[messaggio.length()];
30          int[] intchiave = new int[chiave.length()];
31
32          for (int i = 0; i < messaggio.length(); i++) {
33              intmessaggio[i] = (int) messaggio.charAt(i) - 64;
34          }
35
36          for (int i = 0; i < chiave.length(); i++) {
37              intchiave[i] = (int) chiave.charAt(i) - 64;
38          }
39
40          int[] intcifrato = new int[messaggio.length()];
41
42          String cifrato = "";
43
44          for (int i = 0; i < messaggio.length(); i++) {
45              intcifrato[i] = ((intmessaggio[i] + intchiave[i]) % 26) + 63;
46              cifrato += (char) intcifrato[i];
47          }
48
49          System.out.println("MESSAGGIO CIFRATO: " + cifrato);
50
51      }
52
53  }
```

Il problema principale di questi due cifrari è quello di generare una chiave che sia assolutamente casuale. Come per una moderna *password*, se la chiave è prevedibile o è direttamente o indirettamente legata al contenuto del messaggio, al mittente o al destinatario, essa potrebbe essere scoperta e permettere la decrittazione del messaggio da un utente esterno (si tratterebbe di un attacco *Man-In-The-Middle*, MITM).

3 Crittosistemi moderni

In questo capitolo passeremo in rassegna alcuni dei più recenti crittosistemi utilizzati soprattutto per crittografare informazioni online.

3.1 Crittografia a chiave privata – a chiave pubblica

Quando abbiamo parlato dei crittosistemi antichi, abbiamo dedicato una piccola porzione di questo scritto alla differenza tra crittografia monoalfabetica e crittografia polialfabetica. Mi sembra ora doveroso riprodurre quanto fatto, con le dovute proporzioni, per distinguere fra crittografia **a chiave privata** e crittografia **a chiave pubblica**. Va immediatamente detto che i crittosistemi moderni utilizzano quest'ultimo principio a scapito del primo, più datato. Il motivo sta nel fatto che con la crittografia a chiave pubblica le chiavi non devono essere scambiate attraverso un canale non sicuro, caratteristica essenziale per l'implementazione in Rete.

La crittografia a chiave privata (o *crittografia simmetrica*) è quel tipo di crittografia nel quale viene utilizzata una sola chiave sia per cifrare che per decifrare. È necessario che la chiave sia conosciuta sia al mittente sia al destinatario del messaggio e dev'essere obbligatoriamente mantenuta segreta. I vantaggi offerti da questo tipo di algoritmi sono essenzialmente due: le chiavi possono essere anche molto lunghe, dal momento che l'operazione di cifratura è molto veloce; è inoltre facilmente comprensibile che questi crittosistemi siano i più semplici da implementare, poiché richiedono una sola chiave che viene scambiata fra mittente e destinatario. Si evidenziano però alcuni importanti svantaggi: ad esempio il fatto che un'eventuale utilizzazione multipla di una stessa chiave comporterebbe una falla di sicurezza; esiste inoltre una difficoltà insita nel creare un canale sicuro per la trasmissione della chiave. Questi algoritmi sono difficili da utilizzare in Internet, dato l'immenso bacino di utenti. È infatti facile osservare che se in gioco ci sono n utenti, allora bisogna generare $\frac{n(n-1)}{2}$ chiavi. Per fare un esempio, citiamo la funzione di ricerca di Google. Il sito `www.google.com` è stato visitato più di 81 miliardi di volte nel solo mese di luglio 2020 [16]. Ciò significa che si sarebbero dovute generare circa $3,3 \cdot 10^{21}$ chiavi differenti. A differenza della crittografia simmetrica, la crittografia a chiave pubblica (o *crittografia asimmetrica*) sottende l'implementazione di algoritmi generalmente più complessi e l'utilizzo di chiavi diverse (privata e pubblica). In questo genere di crittosistemi si evita in qualsiasi modo lo scambio non sicuro di un'unica chiave come accade nella crittografia simmetrica, poiché una chiave cifra il messaggio e l'altra lo decifra. Nella figura seguente è schematizzato uno scambio di corrispondenza elettronica tra Alice e Bob utilizzando la crittografia asimmetrica.

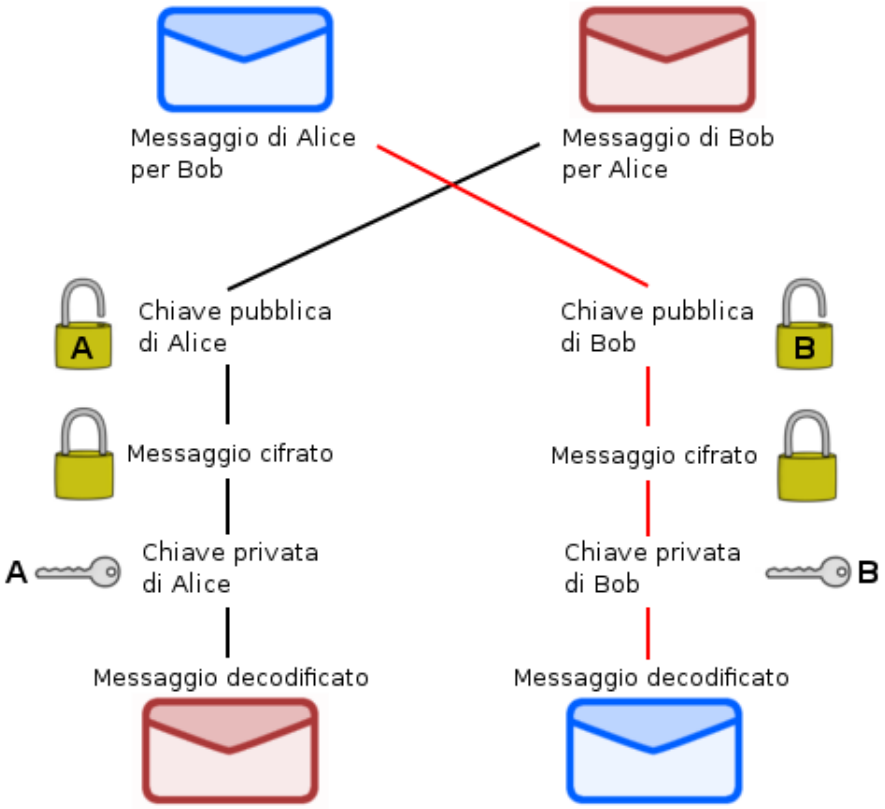

Figura 1: Schema di uno scambio di corrispondenza elettronica criptato con crittografia asimmetrica — [9].

Nella figura precedente si vede bene come funziona la crittografia asimmetrica. Analizziamo solo l'invio di un messaggio da Alice a Bob; va da sé che l'invio inverso funziona allo stesso modo.

1. Alice manda un messaggio a Bob chiudendo il lucchetto (cioè applicando la funzione crittografica) con la chiave pubblica di Bob.

2. Il messaggio viene trasmesso con il lucchetto chiuso.

3. Bob può aprire il lucchetto (cioè decrittare) con la sua chiave privata.

La chiave pubblica è messa a disposizione da ogni utente verso tutti gli altri; al contrario, la chiave privata non va mai trasmessa e serve per decrittare.

3.2 L'algoritmo Diffie-Hellman

Lo scambio di chiavi Diffie-Hellman è un algoritmo crittografico che permette a due entità, negli esempi seguenti Alice e Bob, di concordare una chiave solo a loro conosciuta utilizzando tuttavia un canale pubblico, che è insicuro. Il motivo dell'insicurezza è essenzialmente il fatto che un canale pubblico è esposto a tutti gli altri utenti ed è quindi attaccabile facilmente. La chiave concordata può essere utilizzata, per esempio, in algoritmi crittografici simmetrici. La sua sicurezza è basata sull'elevata complessità computazionale del calcolo del logaritmo discreto.

Questo algoritmo è stato inventato nel 1976 grazie a una collaborazione tra gli statunitensi Bailey Whitfield DIFFIE, crittografo e Martin Edward HELLMAN, crittografo e informatico.

Nonostante la data di invenzione di questo algoritmo faccia ancora parte del XX secolo, ho deciso di includerlo nella sezione dedicata ai crittosistemi recenti, poiché ancora oggi si fa grande uso dello scambio di chiavi Diffie-Hellman.

Di seguito è spiegato ed esemplificato il funzionamento dell'algoritmo.

1. Alice genera e rende pubblico un numero **primo** N molto grande (ad es. di 1024 bit) e un *generatore* g. Siccome N è primo per definizione, allora g esiste sicuramente.
2. Alice genera un numero causale $a < N$ e calcola $A \equiv g^a \pmod{N}$. A viene reso pubblico, mentre a è conosciuto solo ad Alice.
3. Allo stesso modo, Bob genera un numero casuale $b < N$ che mantiene privato e pubblica $B \equiv g^b \pmod{N}$.
4. Ora, Alice calcola $k \equiv B^a \pmod{N}$ e Bob calcola $k \equiv A^b \pmod{N}$.
5. Si ottengono due numeri uguali, che valgono $k \equiv g^{ab} \pmod{N}$.
6. È ora possibile utilizzare k in un crittosistema simmetrico.

Propongo un esempio numerico:

1. Alice genera $N = 5$ e $g = 3$.
2. Alice sceglie un numero $a = 2$ che è minore di 5.
3. Bob sceglie a sua volta un $b = 3$ minore di 5.
4. Alice pubblica un $A \equiv g^a \pmod{N}$ e ottiene $A = 3^2 = 9 \equiv 4 \pmod{5}$.
5. Lo stesso lavoro è svolto da Bob che pubblica $B = 3^3 = 27 \equiv 2 \pmod{5}$.

6. Entrambi i membri calcolano $k \equiv B^a \pmod{N}$ e, rispettivamente, $k \equiv A^b \pmod{N}$. Sia Alice che Bob ottengono $\boxed{k = 4}$. Il valore 4 è stato convenuto dai due senza che vi sia stato scambio insicuro di chiavi. Il fine è stato raggiunto.

Di seguito un esempio di codice Python per lo scambio di chiavi Diffie-Hellman. Si noti che N non è grande quanto dovrebbe essere.

```python
#Diffie-Hellman
import random


def isPrime(n):
    if (n == 1):
        return False
    for i in range(2, int((n**0.5 + 1))):
       if(n % i == 0):
           return False
    else:
            return True


N = random.randint(2,30)
while(isPrime(N) == False):
    N = random.randint(2,30)

#Array dei generatori di N
g2 = [1]
g3 = [2]
g5 = [2,3]
g7 = [3,5]
g11 = [2,6,7,8]
g13 = [2,6,7,11]
g17 = [3, 5, 6, 7, 10, 11, 12, 14]
g19 = [2, 3, 10, 13, 14, 15]
g23 = [5, 7, 10, 11, 14, 15, 17, 19, 20, 21]
g29 = [2, 3, 8, 10, 11, 14, 15, 18, 19, 21, 26, 27]

gN = []

if(N == 2):
    gN = g2
```

```
35 elif(N == 3):
36     gN = g3
37 elif(N==5):
38     gN = g5
39 elif(N==7):
40     gN = g7
41 elif(N==11):
42     gN = g11
43 elif(N==13):
44     gN = g13
45 elif(N==17):
46     gN = g17
47 elif(N==19):
48     gN = g19
49 elif(N==23):
50     gN = g23
51 elif(N==29):
52     gN = g29
53 else:
54     exit()
55
56 # Scelta di g
57 j = random.randint(0, len(gN)-1)
58 g = gN[j]
59
60 #Valori a, b casuali, privati, a,b < N
61
62
63 a = random.randint(2,N)
64 b = random.randint(2,N)
65
66 while(a == b): #Non voglio che a = b
67     a = random.randint(2,N)
68     b = random.randint(2,N)
69
70 #Calcolo di A = g^a (mod N) e B = g^b (mod N)
71 A = (g**a)%N
72 B = (g**b)%N
73
74
```

```
75
76 #Calcolo di kA = A^b (mod N) e kB = B^a (mod N), confronto (devono essere
    ↪ uguali)
77 kA = (A**b)%N
78 kB = (B**a)%N
79 if(kA == kB):
80     k = kA
81 else:
82     print("Errore generale.")
83
84 if(k !=((g**(a*b))%N)):
85     print("Errore generale.")
86 else:
87     print("N = " + str(N) + "\t g = " + str(g) + "\t a = " + str(a) + "\t b =
     ↪ " + str(b) + "\t A = " + str(A) + "\t B = " + str(B) + "\n k = " +
     ↪ str(k))
```

L'algoritmo Diffie-Hellman è insicuro dal punto di vista di un attacco *Man-In-The-Middle*. È l'unico attacco per cui l'algoritmo non offre protezione, poiché se è vero che l'attaccante sarebbe impossibilitato, nella pratica, a calcolare $k = g^{ab} \pmod{N}$ non conoscendo né a né b, egli potrebbe tuttavia interporsi fra Alice e Bob e fingere. Se Alice sta scambiando messaggi con qualcuno ma non sa chi è, l'attaccante potrebbe facilmente fingersi il legittimo destinatario. In realtà, matematicamente parlando, il MITM non è l'unico attacco possibile, in quanto sarebbe sufficiente calcolare: $a = \log_g A \pmod{N}$ e $b = \log_g B \pmod{N}$, ma il calcolo del *logaritmo discreto*, il corrispettivo del logaritmo ordinario nell'aritmetica modulare, è computazionalmente troppo oneroso per numeri di 1024 bit e oltre.

Questo è un tipico esempio di ciò che spesso accade in crittografia: un algoritmo crittografico non è *assolutamente sicuro*, nel senso che da un punto di vista teorico esiste la possibilità di riuscire in un attacco. La pratica, però, è ben diversa: il tempo e/o la complessità del calcolo necessario per attaccare l'algoritmo sono troppo elevati, scoraggiando un potenziale attaccante. Spesso l'informazione che è stata criptata serve all'attaccante in breve tempo: se riesce ad ottenerla dopo 20 anni di lavoro, l'operazione perde di significato.

3.3 L'AES — Advanced Encryption Standard

3.3.1 Introduzione e prerequisiti

In questo capitolo parliamo di un algoritmo crittografico simmetrico che si è guadagnato la fama di essere l'algoritmo di sicurezza scelto dal governo degli Stati Uniti, più precisamente dal National Institute of Standards and Technology (NIST), a partire dal 2001. È stato progettato da due crittografi belgi, Joan DAEMEN (1965 – *) e Vincent RIJMEN (1970 – *) ed ha sostituito il crittosistema DES, Data Encryption Standard, considerato non più sicuro dallo stesso NIST. L'AES è un'evoluzione del DES, che però non verrà trattato in questo testo. Il NIST pose i requisti al nuovo algoritmo, che fu proposto nell'ambito di un concorso pubblico. I requisiti erano [5]:

1. il crittosistema doveva essere a chiave simmetrica e di tipo *a pacchetto*[3] (come ad esempio il crittosistema di Hill);
2. la lunghezza della chiave doveva essere compresa tra 128 e 256 bit;
3. le risorse di calcolo dovevano essere limitate per consentire l'implementazione su *smart-card*[4].

Il *Rijandel*, nome dell'algoritmo prima che ricevesse la denominazione ufficiale di AES, vinse il concorso fra 15 algoritmi proposti.

3.3.2 Descrizione del funzionamento

L'AES si compone di quattro operazioni fondamentali, tutte funzioni biiettive:

1. espansione E;
2. sostituzione S;
3. spostamento righe SR;
4. mescolamento colonne MC.

Il fatto che tutte le funzioni siano biiettive permette la decifratura di AES, poiché per ogni funzione è calcolabile la sua inversa.

Esistono essenzialmente tre versioni di AES, che differiscono per la lunghezza del pacchetto di dati da crittografare: si possono trovare AES128, AES192 e AES256, dove il numero rappresenta la lunghezza del pacchetto. Qui tratteremo solo AES
128, ma ogni considerazione fatta rimane valida in tutti i casi.

[3] Corsivo mio.

[4] Una *smart-card* è una tessera magnetica contenente un chip che viene attivato quando entra in contatto con un apposito lettore che gli trasmette la quantità di energia necessaria per inviargli piccoli pacchetti di dati.

Non intendo esporre tutta la teoria che servirebbe per chiarire esattamente il funzionamento dell'AES, anche perché servirebbero dei prerequisiti che esulano dagli scopi di questo Lavoro. Per chi desiderasse approfondire, consiglio la lettura di [5], pp. 36-42 e di [4], pp. 223-225. Espongo solamente un diagramma di flusso che ci aiuterà nella comprensione del successivo esempio, il quale utilizza una «versione semplificata» di AES, ove il processo è iterato solo una volta invece delle canoniche 11 previste per AES-128.

Procedo quindi con un semplice

Esempio. Si ammetta di voler crittografare il messaggio in chiaro $m =$ «`the quick brown fox jumps over the lazy dog`» con AES. In questo caso, utilizziamo un pacchetto dati di lunghezza $l = 128$ bit. Siccome ci sono 8 bit in un byte, possiamo scrivere m in una matrice 4×4 ($16 \cdot 8 = 128$).

0. **Operazioni preliminari.** Convertiamo innanzitutto m in base esadecimale. Questo permette il calcolo da parte del computer ed ha il pregio di avere una rappresentazione più semplice rispetto al codice binario (cioè numeri in base 2). Per la conversione, passiamo dalla codifica più semplice, la ASCII (acronimo per *American Standard Code for Information Interchange*), e in particolare dalla tabella ASCII, che fa corrispondere ad ogni carattere un numero (binario o esadecimale). Tale tabella è consultabile su `https://ascii.cl/`.

   Nota. Il sistema esadecimale fa uso della base

   $$\mathcal{B}_{16} = \{0, 1, \dots, 9, A, B, C, D, E, F\}$$

   Così il numero $(140)_{10}$ corrisponde al numero $(8C)_{16}$.

   Abbiamo pertanto

   $$(m)_{16} = \begin{pmatrix} 74 & 68 & 65 & 20 \\ 71 & 75 & 69 & 63 \\ 6B & 20 & 62 & 72 \\ 6F & 77 & 6E & 20 \\ 66 & 6F & 78 & 20 \\ 6A & 75 & 6D & 70 \\ 73 & 20 & 6F & 76 \\ 65 & 72 & 20 & 74 \\ 68 & 65 & 20 & 6C \\ 61 & 7A & 79 & 20 \\ 64 & 6F & 67 & 2E \end{pmatrix}$$

Per semplificare, tratteremo solo del primo blocco di 16 caratteri; costruiremo quindi la matrice

$$(m')_{16} = \begin{pmatrix} 74 & 68 & 65 & 20 \\ 71 & 75 & 69 & 63 \\ 6B & 20 & 62 & 72 \\ 6F & 77 & 6E & 20 \end{pmatrix}$$

Occupiamoci ora delle chiavi. Scegliamo una chiave k lunga quanto m', per esempio

$$k = \begin{pmatrix} 71 & 34 & 74 & 37 \\ 77 & 21 & 7A & 25 \\ 43 & 2A & 46 & 2D \\ 4A & 61 & 4E & 64 \end{pmatrix}$$

1. **Espansione.** Adesso dobbiamo espandere la chiave. In AES, la funzione $E(k)$ produce, a partire da una sola chiave k un vettore di 11 righe

$$E(k) = \begin{pmatrix} K_0 \\ \vdots \\ K_{10} \end{pmatrix}$$

Per non appesantire questo esempio, attueremo una sola iterazione. L'espansione di AES prevede quattro sotto-operazioni. Non descrivo tali operazioni, perché i prerequisiti necessari per comprenderle esulano da questo lavoro. Ci basti sapere che, per il k dato sopra, è $E(k) = K$, perché svolgiamo una sola iterazione. Al crescere del numero di iterazioni, $E(k)$ produce il vettore di cui sopra.

$$K = E(k) = \begin{pmatrix} 71 & 34 & 74 & 37 \\ 77 & 21 & 7A & 25 \\ 43 & 2A & 46 & 2D \\ 4A & 61 & 4E & 64 \\ 9F & 1B & 37 & E1 \\ E8 & 3A & 4D & C4 \\ AB & 10 & 0B & E9 \\ E1 & 71 & 45 & 8D \end{pmatrix}$$

Lo scopo di questa funzione è quella di generare facilmente delle *chiavi di ciclo*. Maggiore è il numero di iterazioni, maggiore è la sicurezza di AES. Per convenzione, AES-128 usa 10 iterazioni, AES-192 usa 12 iterazioni e AES-256 usa 14 iterazioni.

2. **Sostituzione.** La seconda operazione è l'addizione binaria (XOR, $\oplus$) della chiave K_i al messaggio in chiaro (o, meglio, allo *stato*, dato che dopo la prima iterazione non si ha più il messaggio in chiaro, ma un valore aggiornato dalle iterazioni precedenti). Siccome qui abbiamo una sola iterazione ($i = 1$), effettuiamo semplicemente $s = m' \oplus K$. Dalla Definizione 1.9, apprendiamo che i valori devono prima esser convertiti in binario, addizionati e riconvertiti in esadecimale.

3. **Spostamento righe.** Lo spostamento righe agisce nel seguente modo: sposta la seconda riga a sinistra di una posizione, la terza di due e la quarta di tre. Se lo stato è
$$s = \begin{pmatrix} s_0 & s_4 & s_8 & s_{12} \\ s_1 & s_5 & s_9 & s_{13} \\ s_2 & s_6 & s_{10} & s_{14} \\ s_3 & s_7 & s_{11} & s_{15} \end{pmatrix}$$
allora
$$SR(s) = \begin{pmatrix} s_0 & s_4 & s_8 & s_{12} \\ s_5 & s_9 & s_{13} & s_1 \\ s_{10} & s_{14} & s_2 & s_6 \\ s_{15} & s_3 & s_7 & s_{11} \end{pmatrix}$$

4. **Mescolamento colonne.** La funzione MC viene definita come operazione su quaterne di byte:
$$MC \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix}_{16} \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{pmatrix}$$

5. **Conclusione.** Infine, il nostro blocco m' risulta in
$$c' = \begin{pmatrix} F4 & 3B & 01 & FA \\ 87 & 5D & FA & 34 \\ 9F & 57 & 89 & B3 \\ DE & 3B & 38 & 42 \end{pmatrix}$$

Per chi desiderasse conoscere praticamente il funzionamento di AES, consiglio di visitare la pagina [14] che consente di testare il funzionamento di AES sia in configurazioni standard, sia personalizzando ogni aspetto del crittosistema (numero di iterazioni, chiave principale, messaggio in chiaro, ...) e permette di mostrare il risultato intermedio dopo ogni fase e ogni iterazione. Presento infine un diagramma di flusso per AES-128.

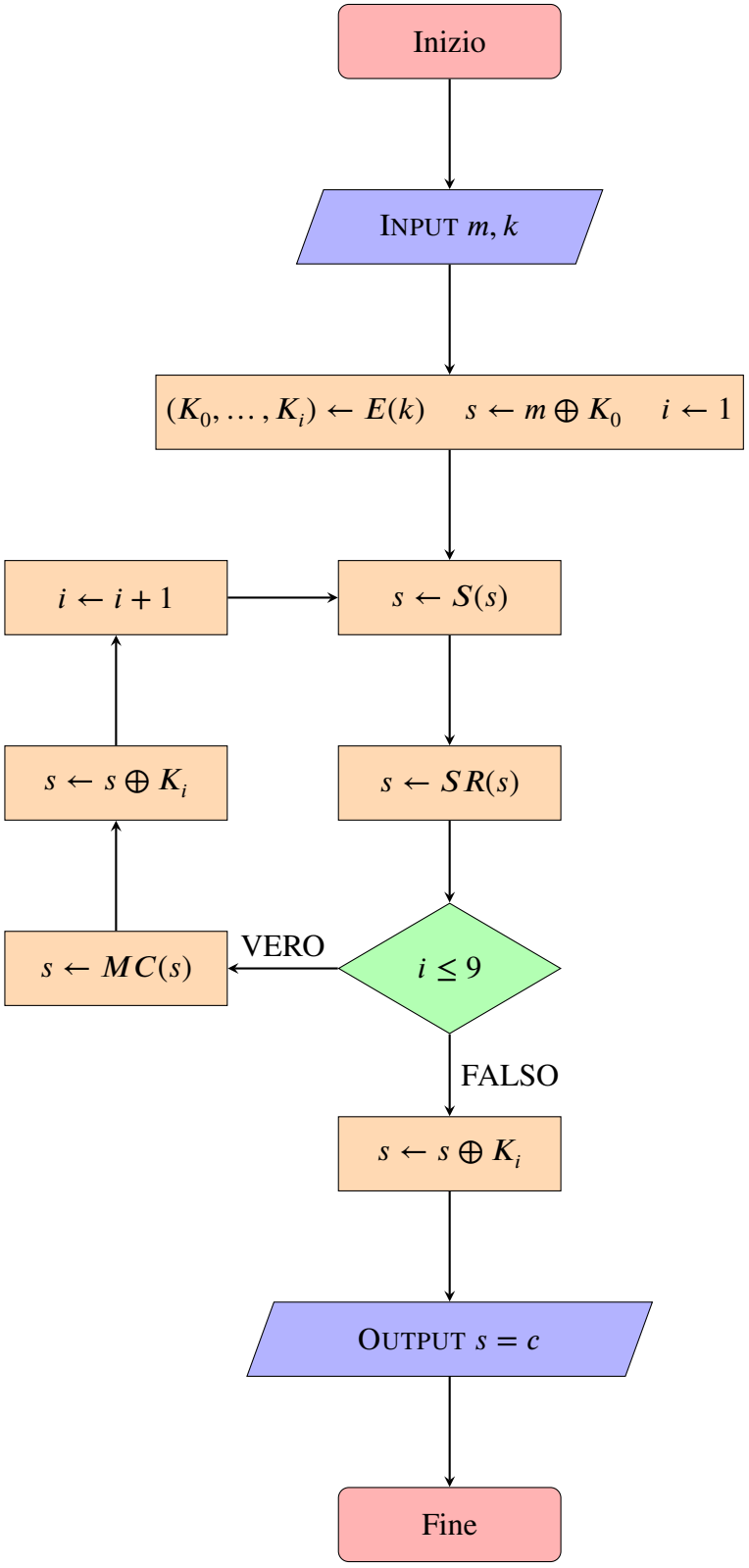

Figura 2: diagramma di flusso per AES-128. Riadattato da [5], p. 37.

3.3.3 Decifratura di AES

La decifratura di AES è garantita dal fatto che tutte le quattro funzioni coinvolte sono invertibili. Per decifrare è necessario svolgere le quattro operazioni indicate in ordine inverso, usando come chiavi di ciclo quelle contenute nel vettore

$$K' = \begin{pmatrix} K_{10} \\ \vdots \\ K_0 \end{pmatrix}$$

cioè le stesse del vettore K prese però in ordine inverso. Le operazioni SR, S, MC vanno sostituite con le loro inverse: l'operazione di somma bit a bit è identica alla sua inversa, S^{-1} si ottiene con delle traslazioni a destra (invece che a sinistra) e MC^{-1} è definita dalla matrice inversa rispetto a quella data precedentemente, ossia:

$$\begin{pmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{pmatrix}$$

3.4 L'RSA

3.4.1 Breve storia

L'algoritmo crittografico **RSA** è un crittosistema asimmetrico nato nel 1978 dalle menti di Ronald RIVEST, crittografo statunitense, Adi SHAMIR, crittografo, informatico e matematico israeliano e Leonard ADLEMAN, matematico, informatico e biologo statunitense. Il nome dell'algoritmo è derivato dall'iniziale del cognome di questi tre scienziati. È diventato lo standard per la crittografia asimmetrica ed è stato formulato appena un anno dopo la pubblicazione del metodo di scambio di chiavi Diffie-Hellman, da cui in parte trae la filosofia. È stato brevettato nel 1983 dal MIT.

3.4.2 Funzionamento

In questa sezione descrivo praticamente il funzionamento dell'RSA, senza entrare nei particolari matematici; a questi sarà dedicata la prossima ampia sezione.

Ogni utente in gioco, qui chiamato Alice come spesso accade in letteratura, svolge le seguenti operazioni una volta sola:

1. Alice sceglie due numeri **primi** p e q «sufficientemente grandi» (si veda la prossima sezione per capire cosa si intende con questo vago termine; l'ordine di grandezza raccomandato dagli RSA Laboratories è di 2048 bit, cioè 617 cifre decimali [12]);

2. Alice calcola $n = pq$, fermo restando che la fattorizzazione di n rimane conosciuta alla sola Alice;
3. Alice calcola $\varphi(n) = (p-1)(q-1)$;
4. Alice sceglie un numero $e \in \mathbb{N} : (e, \varphi(n)) = 1$[5];
5. Alice calcola $d \in \mathbb{Z}^*_{\varphi(n)} : e \cdot d \equiv 1 \pmod{\varphi(n)}$;
6. Alice rende nota la coppia (n, e), detta **chiave pubblica** e mantiene segreta la terna (p, q, d) che è la sua **chiave privata**.

Il vantaggio di questa procedura è che la sua complessità computazionale è al massimo polinomiale ed è quindi facilmente gestibile dai moderni computer e dispositivi elettronici vari.

Per crittografare un messaggio è necessario applicare la funzione f così definita:

Definizione 3.1 (Funzione crittografica dell'RSA)**.**
La funzione crittografica dell'utente generico è definita come:

$$\begin{aligned} f : \mathfrak{M} &\to \mathfrak{C} \\ x &\mapsto y = f(x) = x^e \pmod{n} \end{aligned}$$

Come si vede, tutti gli utenti del crittosistema possono codificare un messaggio perché f contiene solo parametri della chiave pubblica. Per decifrare è definita, per ogni utente del crittosistema, la sua personale funzione di decrittazione che fa uso della sua chiave privata.

Definizione 3.2 (Funzione di decrittazione dell'RSA)**.**
Sia data la **funzione di decrittazione** dell'RSA così definita:

$$\begin{aligned} f^{-1} : \mathfrak{C} &\to \mathfrak{M} \\ y &\mapsto f^{-1}(y) = x = y^d \pmod{n} \end{aligned}$$

Come si vede, per decifrare è necessario conoscere d e quindi $\varphi(n)$ e quindi la fattorizzazione di n, cioè p e q.

Esempio. Bob desidera codificare con RSA il seguente messaggio da destinare ad Alice: $m =$ «`The quick brown fox jumps over the lazy dog.`», 44 caratteri, spazi e punteggiatura compresi. Per far corrispondere ogni blocco «`ABCD`» da 4 caratteri a un numero, usiamo il polinomio

$$p(\texttt{ABCD}) = \texttt{A} \cdot 28^3 + \texttt{B} \cdot 28^2 + \texttt{C} \cdot 28 + \texttt{D}$$

[5] Con questa notazione si intende MCD $(e, \varphi(n))$.

che restituisce un numero in base 28, e consideriamo le lettere dell'alfabeto inglese come numeri da `A` $= 0$ a `Z` $= 25$, «␣» $= 26$ e «.» $= 27$.

È importante notare che questa non è un'applicazione standard di RSA; si tratta piuttosto di una «versione didattica» dell'RSA, dove lavoriamo con numeri il cui ordine di grandezza è molto minore.

Scriviamo la Tabella 2 ove sono calcolati i numeri corrispettivi ad ogni carattere di m, il valore $p(M)$, dove M è un gruppo di 4 caratteri consecutivi di m e il valore della funzione $f(p(M)) = p(M)^e \pmod{n}$. Abbiamo usato per crittografare la chiave pubblica di Alice $(n, e) = (2535311, 10001)$. Alice sarà in grado di decifrarlo usando la sua chiave privata (p, q, d). Bob trasmetterà ad Alice il seguente messaggio criptato:

$$c = 808231, 1438012, 844705, 1688101, 468290, 1375232, 574189, 554629, 1533131,$$

$$2381712, 1071163$$

Carattere di m	**Numero corrispettivo**	$p(M)$	$p(M)^e \pmod n$
T	19	422714	808231
H	7		
E	4		
␣	26		
Q	16	367138	1438012
U	20		
I	8		
C	2		
K	10	239949	844705
␣	26		
B	1		
R	17		
O	14	324966	1688101
W	22		
N	13		
␣	26		
F	5	134412	468290
O	14		
X	23		
␣	26		
J	9	236684	1375232
U	20		
M	12		
P	15		
S	18	461295	574189
␣	26		
O	14		
V	21		
E	4	112626	554629
R	17		
␣	26		
T	19		
H	7	174852	1533131
E	4		
␣	26		
L	11		
A	0	21747	2381712
Z	25		
Y	24		
␣	26		
D	3	85142	1071163
O	14		
G	6		
.	27		

Tabella 2: Tabella contenente i caratteri di m, il loro corrispettivo numerico, il valore $p(M)$ e il valore $f(p(M))$.

3.4.3 Basi matematiche

Dedichiamo qui un'ampia sezione alla motivazione matematica del funzionamento dell'algoritmo RSA e lo faremo in modo sistematico, passo per passo, seguendo la procedura data nella sezione precedente.

1. *Scegliere due numeri primi p, q sufficientemente grandi.* Nell'esempio mostrato poc'anzi si poteva effettivamente «scegliere»; nella realtà, la scelta non è completamente libera. Il problema è infatti posto da quel «sufficientemente grandi». Esistono i cosidetti **Numeri RSA**, dei numeri semiprimi (cioè composti da due fattori primi distinti) che sono usati come valore di n. Possiamo verificare su [12] che il numero RSA-250, il più grande fattorizzato fin ora, possiede 250 cifre decimali ed è composto da due primi di 120 cifre ciascuno.

   Il fatto che p, q siano primi rende pure facile il calcolo di $\varphi(n)$, siccome vale: $\varphi(n) = \varphi(pq) = \varphi(p) \cdot \varphi(q) = (p-1)(q-1)$. Questo fatto discende direttamente da due importanti considerazioni su φ:

   - $\varphi(ab) = \varphi(a) \cdot \varphi(b) \quad \forall a, b$;
   - $\varphi(p) = p - 1 \Leftrightarrow p$ è primo.

   Bisogna notare che è altamente sconsigliato crittografare qualsiasi informazione con una versione di RSA che fa uso di un numero n già fattorizzato: questo inficia infatti l'algoritmo, perché se n è conosciuto ad un potenziale attaccante, allora questi può calcolare facilmente $\varphi(n)$. Dato che già conosce p e q, allora è in grado di determinare d se scopre e (si veda il punto *Calcolo di d*) e pertanto di decifrare un messaggio in luogo del suo legittimo destinatario.

2. *Calcolare $n = pq$.* La sicurezza del crittosistema RSA si basa proprio sul fatto che dev'essere tenuta segreta dal mittente la fattorizzazione di n, cioè p e q per i motivi visti sopra.

3. *Calcolare $\varphi(n)$.* La potenza dell'algoritmo RSA sta proprio in questo cruciale passaggio. Sappiamo che la chiave pubblica è della forma (n, e), mentre quella privata è della forma (p, q, d); per passare da e a d (o viceversa) non è sufficiente conoscere n, bensì serve $\varphi(n)$. Il calcolo di questo valore è computazionalmente poco oneroso se si conoscono i fattori di n, ma necessita di tempi molto lunghi se va usata la formula data nella definizione di questa funzione: si tratta infatti di una produttoria che richiede di conoscere i fattori di n.

4. *Scelta di e.* La scelta di e non riveste un ruolo fondamentale nella riuscita e nella sicurezza di RSA. Come si legge in [4], p. 313: «Non sembra esserci differenza nella difficoltà del problema RSA [cioè quello di fattorizzare n] per differenti esponenti e e, pertanto, sono stati proposti vari metodi per selezionarlo. Una scel-

ta popolare è $e = 3$, dal momento che il calcolo delle e-esime potenze modulo n richiede solo due moltiplicazioni. Se e è scelto pari a 3, allora p e q devono essere scelti con $p, q \not\equiv 1 \pmod 3$, così che $(e, \varphi(n)) = 1$. Per ragioni analoghe, un'altra scelta comune è $e = 2^{16} + 1 = 65537$, un numero primo con un basso *peso di Hamming*[6] [...] Rispetto a $e = 3$, questo valore rende l'elevazione a potenza più costosa, ma riduce le limitazioni circa la scelta di p, q e evita alcuni 'attacchi da basso esponente' che possono derivare da una cattiva implementazione di un cifrario basato su RSA».

5. *Calcolo di d.* La conoscenza di d è fondamentale, perché è il valore che permette la corretta decifrazione del messaggio. Se un attaccante volesse decifrare un messaggio non a lui indirizzato, dovrebbe calcolare d sapendo che $d \equiv e^{-1} \pmod{\varphi(n)}$, ma per farlo deve conoscere $\varphi(n)$ e quindi i fattori di n.

3.4.4 Codice Python — Crittazione

Il seguente codice Python3 esegue la crittazione di una stringa, generando p e q casualmente tra 2 e 2^{32} e ponendo e un numero casuale tra 3 e $2^{16} + 1$ per i motivi detti sopra (punto 4. – *Scelta di e*). Questo permette di mantenere il tempo di esecuzione accettabile per un programma esemplificativo e non adatto alla produzione.

```
1  #RSA
2  import random
3  import math
4  # Inizializzazione variabili
5  p = 0
6  q = 0
7  n = 0
8  phi = 0
9  e = 0
10 d = 0
11 candidates = []
12
13 def genNum():
14     p = random.randint(2, 2**32)
15     q = random.randint(2, 2**32)
16     candidates.append(p)
```

[6] Il **peso di Hamming** di un numero è il numero di elementi 1 presenti nella sua rappresentazione binaria. Più questo valore è basso, più veloce sarà l'elevazione a potenza richiesta in f. Questa definizione può apparire lontana dai nostri scopi; si tengano però presenti le implicazioni di questo valore. Per approfondire, si veda [4], pp. 553 – 556.

```
17      candidates.append(q)
18      return candidates
19
20  def isPrime(n):
21      if (n == 1):
22          return False
23      for i in range(2, int((n**0.5 + 1))):
24         if(n % i == 0):
25             return False
26      else:
27              return True
28
29  #1. Scelta (casuale) di p e q.
30
31  print("Attendere mentre vengono generati i numeri p e q...\n")
32  genNum()
33  while(isPrime(candidates[0]) == False or isPrime(candidates[1]) == False or
     ↪  candidates[0] == candidates[1]):
34      candidates = []
35      genNum()
36  print("I numeri p, q sono: " + str(candidates))
37
38  #2. Calcolo di n = pq
39  p = candidates[0]
40  q = candidates[1]
41  n = p*q
42
43  #3. Calcolo di phi(n) = (p-1)(q-1)
44  phi = (p-1)*(q-1)
45  print("phi(n) = " + str(phi))
46
47  #4 Scelta di e
48  #e è compreso tra 3 e 2^16 -1, così da contenere il tempo di esecuzione.
49  while(math.gcd(e,phi)!=1):
50      e = random.randint(3,2**16-1)
51  print("e = " + str(e))
52
53
54  #5. Cifratura
55  m = input("\nInserire il messaggio in chiaro da cifrare.\n")
```

```
#m = m.upper() # Uso solo lettere maiuscole per semplicità
caratteri_m = []

for i in range(0, len(m)):
  caratteri_m.append(m[i])

M = [] #Contiene i codici ASCII del corrispettivo carattere di m
for i in range(0, len(m)):
  M.append(ord(m[i]))

C = [] #Contiene i codici ASCII del corrispettivo carattere criptato di m
def cifratura():
  for i in range(0, len(m)):
    c = (M[i]**e) % n
    C.append(c)

cifratura()

print("\n\n Il messaggio cifrato è: ")
print(C)
```

3.4.5 Decrittazione

La decrittazione di un messaggio crittografato con RSA si avvale della funzione f^{-1} definita in 3.2. Per decifrare è necessario conoscere il valore d, presente nella chiave privata del destinatario. Come detto, siccome d è stato scelto tale che: $e \cdot d \equiv 1 \pmod{\varphi(n)}$, allora $d \equiv e^{-1} \pmod{\varphi(n)}$.

Mi sembra ora chiaro, quindi, cosa si intende con *Problema RSA*: è così denominato qualsiasi problema che richieda di cercare la fattorizzazione di un numero intero n semiprimo (cioè prodotto di due primi distinti). Dal grado di difficoltà di questo problema nasce il grado di sicurezza di una certa applicazione particolare. Parallelamente vengono studiati alcuni tipi di attacco RSA che sono originati (spesso) da scelte errate nell'implementazione stessa di un crittosistema. Vado ad esporne alcune nel prossimo sotto-sottocapitolo.

3.4.6 Attacchi a RSA

Alcuni attacchi a RSA sono detti *elementari*, ad esempio quello che deriva dalla scelta di un insieme $\mathfrak{M}$ molto piccolo. Questo comporta che è possibile, in tempi ragionevoli,

perpetrare un attacco di forza bruta per cercare di scoprire quale messaggio in chiaro corrisponde al messaggio cifrato in nostro possesso. Altri attacchi, di tipo probabilistico, si incentrano sulla ricerca dei fattori di n, dato d piccolo.

Alcuni dei problemi che possono rendere un cifrario basato su RSA vulnerabili sono:

- valori di n che hanno fattori primi piccoli;
- valori di n che hanno fattori primi vicini;
- valori di n tali che $n + 1$ o $n - 1$ sono composti da fattori primi piccoli.

Generare numeri che rispettino le condizioni sopraindicate non è computazionalmente più oneroso che generare numeri primi della stessa lunghezza non rispettando i criteri, come è indicato in [5], p. 56.

Non si vogliono valori di n che abbiano fattori primi piccoli o vicini, perché questi n sono facilmente fattorizzabili con il metodo di Fermat. Allo stesso modo, si rifiutano i valori di n tali che $n - 1$ o $n + 1$ sono composti di primi piccoli, perché questi sono fattorizzabili efficientemente dai metodi $p - 1$ di Pollard e $p + 1$. Proprio allo scopo di dare maggiore solidità a queste affermazioni si veda la sezione successiva.

Attacco di tipo *chosen-ciphertext* [5] Un attacco *chosen-ciphertext* è un tipo di attacco in cui l'attaccante conosce la funzione di decifrazione ma non la (o le) chiavi. Lo scopo è quindi quello di risalire alla o alle chiavi. Spieghiamo come funziona: l'attaccante D si intromette nella comunicazione tra A e B. B invia ad A un messaggio cifrato c. Lo scopo di D è scoprire la chiave che permette di risalire a m. Egli sceglie un intero casuale R e richiede ad A la decodifica di $c_1 \equiv R^{e_A} c \pmod{n_A}$. Ammesso che A evada la richiesta, D ottiene la quantità $m_1 \equiv c_1^{d_A} \pmod{n_A}$. In effetti, A non ha motivo di non eseguire la richiesta, perché a causa di R, il messaggio cifrato c_1 non ha relazione con c. La decodifica prosegue così:

$$m_1 R^{-1} \equiv R^{e_A \cdot d_A} \cdot c^{d_A} \cdot R^{-1} \equiv c^{d_A} \equiv m \pmod{n_A}$$

Capiamo quindi che è importante evitare di codificare messaggi o documenti di provenienza sconosciuta con la propria funzione di cifratura.

3.4.7 Ricerca dei fattori di un $n \in \mathbb{N}$

In questa sezione passeremo in rassegna alcuni algoritmi di fattorizzazione di interi $n \in \mathbb{N}$ tali che $n = pq$, con p, q fattori primi distinti, di egual lunghezza l. Ci concentriamo su questo caso, perché è quello che riguarda il crittosistema RSA. Nello scorrere la carrellata, useremo a volte, senza esplicitarlo, l'enunciato del

Teorema 3 (Cinese del resto)**.**
Siano dati $a, b \in \mathbb{Z}^*$, tali che $\text{MCD}(a, b) = 1$. Allora, il sistema

$$\begin{cases} x \equiv k_1 \pmod{a} \\ x \equiv k_2 \pmod{b} \end{cases}$$

ha un'unica soluzione x_0 modulo ab, cioè della forma $x = x_0 + M(ab)$. Il Teorema è estendibile per un qualsiasi sistema di i equazioni in una sola incognita del tipo stabilito sopra; non è quindi caratteristica esclusiva dei sistemi 2×1.

Dimostrazione. [2] Scindo l'enunciato da dimostrare in due condizioni distinte:

(i) il sistema ammette soluzioni;

(ii) tutte le soluzioni del sistema hanno forma $x = x_0 + M(ab)$, $M \in \mathbb{Z}$.

(i) Come siamo soliti fare nei normali sistemi lineari, possiamo procedere per sostituzione, inserendo la soluzione generale della prima congruenza nella seconda, per ottenere:

$$k_1 + \lambda a \equiv k_2 \pmod{b} \quad (\lambda \in \mathbb{Z})$$

Per le proprietà del calcolo modulare, questo significa che $\lambda a \equiv (k_2 - k_1) \pmod{b}$. Questa congruenza in λ ha sicuramente soluzioni siccome $1 | (k_2 - k_1)$, dove $1 = \text{MCD}(a, b)$, dal momento che per una certa congruenza $ax \equiv b \pmod{n}$, con $a, b, n \in \mathbb{Z}, n > 0$, essa ha soluzione se e solo se $\text{MCD}(a, n) | b$.

✓

(ii) Risolviamo ora la congruenza in λ data in (i). La sua soluzione generale è della forma $\lambda = \lambda_0 + \mu b$, dove λ_0 è una soluzione particolare e $\mu \in \mathbb{Z}$. La soluzione generale del sistema, cioè quella che risolve *contemporaneamente* entrambe le congruenze, è della forma:

$$x = k_1 + a(\lambda_0 + \mu b) = (k_1 + a\lambda_0) + \mu ab \quad ✓ \quad ■$$

Questo ci porta a concludere che il Teorema cinese del resto permette di ridurre il sistema sopraindicato nella congruenza

$$x \equiv k_1 + a\lambda_0 \pmod{ab}$$

dove il secondo membro è una soluzione particolare del sistema.

Per usare una notazione vista nei prerequisiti di Teoria dei Gruppi, il Teorema cinese del resto può anche essere formulato così:

$$\mathbb{Z}_n \simeq \mathbb{Z}_p \times \mathbb{Z}_q \quad \text{e} \quad \mathbb{Z}_n^* \simeq \mathbb{Z}_p^* \times \mathbb{Z}_q^*$$

dove il simbolo «×» indica il prodotto cartesiano. Nel caso dell'RSA, il Teorema così formulato ci permette di semplificare i conti: siccome essi vengono fatti modulo n e la dimensione di n è di 1024 bit o superiori, ma p e q sono sensibilmente più piccoli di n, l'isomorfismo di cui nel Teorema rende i calcoli più semplici dal punto di vista computazionale.

Esempio. Poniamo di voler risolvere il sistema di congruenze lineari

$$\begin{cases} x \equiv 6 \pmod 9 \\ x \equiv 0 \pmod 7 \end{cases}$$

Verifichiamo immediatamente che $\text{MCD}(7, 9) = 1$. Il Teorema è applicabile. Deduciamo che la soluzione sarà modulo $7 \cdot 9 = 63$. La soluzione sarà quindi della forma $x \equiv 6 + 9 \cdot \lambda_0 \pmod{63}$. Ma $\lambda_0 = 4$, perché è soluzione di $6 + 4 \cdot 9 = 0 \pmod 7$. Pertanto, le soluzioni del sistema sono della forma:

$$x \equiv 6 + 4 \cdot 9 \pmod{63} \Leftrightarrow x \equiv 42 \pmod{63}$$

Algoritmo di fattorizzazione di Fermat. Si tratta del più semplice (in relazione a quelli che vediamo qui) degli algoritmi di fattorizzazione, e il suo nome si deve al matematico e magistrato francese Pierre DE FERMAT (1601 – 1665). Questo metodo si basa sulla relazione

$$x^2 - y^2 = (x + y)(x - y)$$

prodotto notevole che permette di rappresentare un numero composto come differenza di quadrati. Se possiamo trovare y tale che $n + y^2 = x^2$, allora significa che $(x - y)|n$, cioè $x^2 \equiv y^2 \pmod n$. Questo algoritmo è generalmente poco efficiente, dato che bisogna scorrere tutti gli y da 0 a n e controllare se vale quanto sopra, tranne nei casi in cui i fattori p, q di n sono vicini. In questo caso y è piccolo e l'algoritmo funziona bene anche per numeri grandi. Ecco spiegato il motivo per cui nella sezione dedicata agli attacchi a RSA scrissi che non è opportuno scegliere p e q vicini fra loro oppure piccoli.

Per terminare, rappresento l'algoritmo con un diagramma di flusso (Figura **??**).

Metodo $p - 1$ di Pollard. Per introdurre questo metodo serve un importante Teorema della Teoria dei Numeri: il Piccolo Teorema di Fermat. Enunciamolo:

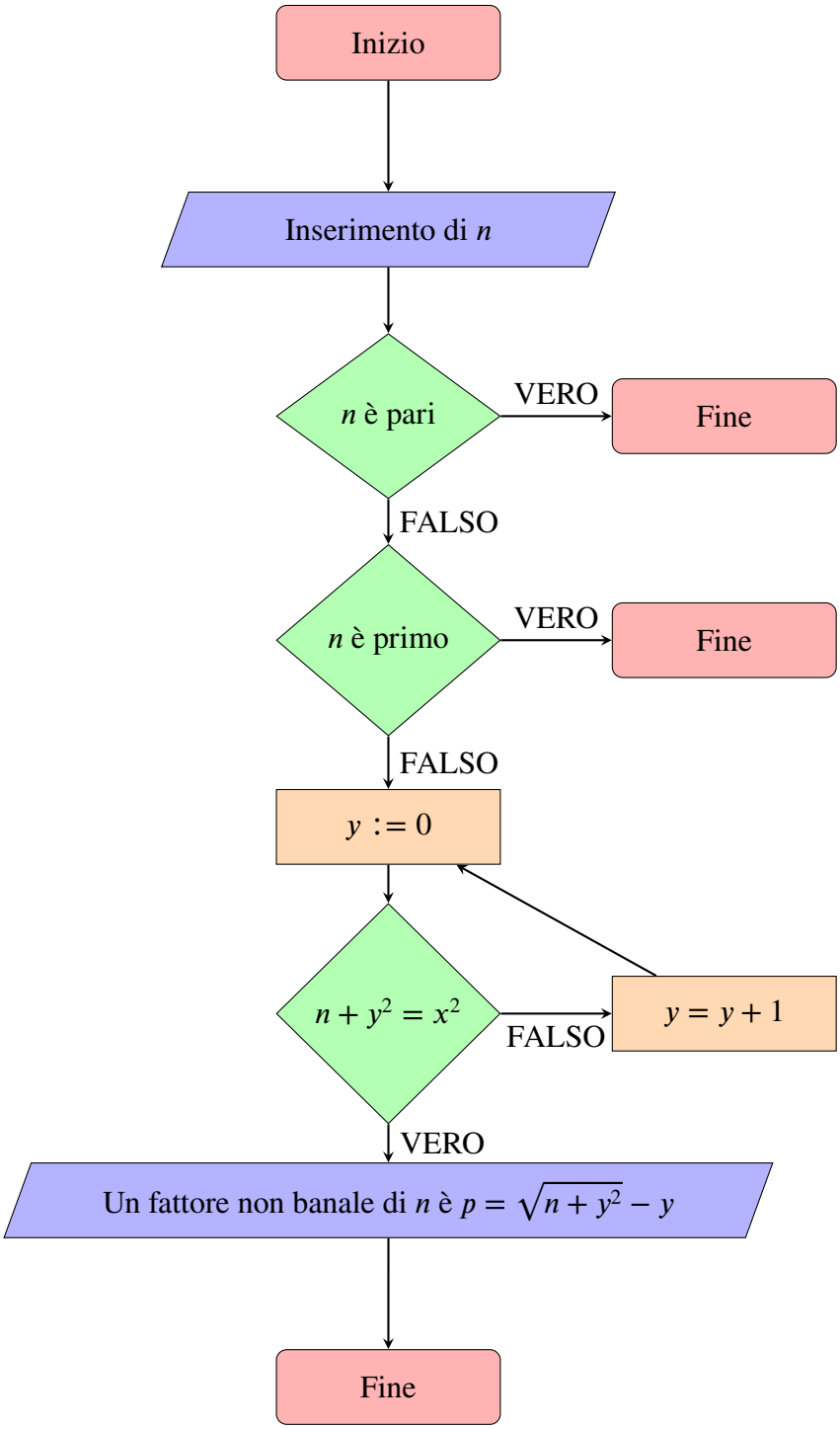

Figura 3: diagramma di flusso del metodo di Fermat

Teorema 4 (Piccolo Teorema di Fermat)**.**
Siano dati $a \in \mathbb{Z}$ e p primo. Allora vale

$$a^p \equiv a \pmod{p}$$

oppure, equivalentemente,

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

ammesso che $\text{MCD}(a, p) = 1$.

Per la dimostrazione di tale Teorema, si veda il [5], p. 249.

Il metodo $p-1$ di Pollard, inventato dal matematico inglese John POLLARD (1941 – *), si basa sull'idea che, dato un naturale composto n e un suo fattore primo p, valga

$$\forall k : p-1|k \quad \text{è} \quad n^k \equiv 1 \pmod{p}$$

che segue direttamente dal Piccolo Teorema di Fermat. Ciò si traduce in due importanti relazioni, ovvero:

$$p|n^k - 1 \quad \text{e} \quad p|n$$

Per trovare un fattore primo di n possiamo ora calcolare $\text{MCD}(n^k - 1, n)$, per ogni k. A prima vista, questo metodo non sembra efficiente e infatti non lo è per ogni n; lo è solo nel particolare caso in cui, per un certo p fattore primo di n, si abbia $p-1$ scomponibile in fattori di piccole dimensioni. In questo caso, k è a sua volta piccolo e, in poche iterazioni, si trova un p non banale. Rimane tuttavia il problema della scelta di un k adeguato: esso deve costituire un limite ragionevole, cioè dev'essere sufficientemente piccolo da non inficiare la già limitata efficienza dell'algoritmo e sufficientemente grande da impedire che l'algoritmo fallisca e restituisca valori banali. Scelte solite di k sono $k = B!$ o $k = \text{mcm}(2, \dots, B)$, dato un certo B fissato in base a quanto detto poc'anzi circa k. In realtà, dietro alla scelta di B sussistono ragionamenti probabilistici («qual è la probabilità di trovare un numero primo in un certo intervallo?») che però non approfondisco in questo lavoro.

4 Applicazioni pratiche degli algoritmi crittografici

In questo breve capitolo conclusivo elenco qualche applicazione pratica dei moderni algoritmi crittografici visti nel capitolo precedente. In alcuni casi, più crittosistemi possono venir utilizzati nello stesso processo crittografico, al fine di garantire un'ancora maggiore sicurezza.

AES

Il crittosistema AES è usato spesso per proteggere le seguenti tecnologie:

Sistemi VoIP I sistemi VoIP, sigla per *Voice over IP*, costituiscono la base per le chiamate instradate via Internet, anziché via rete telefonica (analogica). Tali sistemi sono usati soprattutto nelle aziende per i numeri interni.

Smart card Come richiesto dal NIST, il sistema AES è adatto per essere utilizzato su tessere magnetiche *smart card*, come carte di credito, token di accesso a postazioni PC di lavoro, sblocco di porte, ecc. Questo fatto è spiegato dal relativamente basso costo computazionale di AES. Queste tessere si compongono di un chip che contiene le informazioni essenziali sotto forma di piccoli pacchetti di dati; il chip è alimentato per induzione quando è avvicinato all'apposito lettore (in modo del tutto analogo a quello che avviene con i moderni sistemi di pagamento *contactless*). Le informazioni trasmesse al lettore sono crittografate secondo AES, essendo quindi al riparo da attacchi MITM.

RSA

Il crittosistema RSA è sovente utilizzato nelle comunicazioni che avvengono tramite la rete Internet; in particolare, è alla base della connessione SSL/TLS, che permette di crittografare le informazioni che passano, altrimenti in chiaro, tramite HTTP (*HyperText Transfer Protocol*). In realtà, come vedremo, anche AES entra in gioco in questo processo. Di seguito chiarisco il funzionamento di una connessione criptata a un certo sito web, e come essa differisca da una connessione in chiaro.

Va immediatamente precisato che, se una normale connessione a un sito web si svolge nell'ambito del protocollo HTTP sulla porta 80, una connessione criptata si svolge in HTTPS, protocollo sintatticamente identico al precedente, che opera però sulla porta 443.

Connessione criptata *passo per passo* Un client C desidera connettersi al sito gestito da un server S. Per avviare una connessione protetta, C e S devono svolgere la procedura

descritta qui di seguito, detta *TLS Handshake* (lett. «stretta di mano»).
Il processo è descritto di seguito e rappresentato graficamente dalla Figura 4.

La prima fase è la **negoziazione** tra le parti del crittosistema da utilizzare, la seconda consiste nello **scambio** della **chiave per la crittografia *simmetrica*** (tipicamente AES) per mezzo di un canale crittografato ***asimmetricamente*** (RSA2048) e l'autenticazione tramite l'utilizzo di un certificato. Il certificato serve a garantire che il server è effettivamente chi dice di essere, e non un server gestito, per esempio, da malintenzionati. Capiamo che il problema è solamente traslato all'indietro, e non risolto. Chi ci garantisce, infatti, che il certificato sia *autentico*? A questo ci pensa un ente, detto *Autorità di Certificazione* (*Certification Authority*, CA in inglese) deputato all'emissione e alla verifica dei certificati. A questo punto siamo costretti ad assumere che la CA sia onesta. È un'assunzione fatta per noi dal produttore del dispositivo con cui ci colleghiamo alla rete: egli stabilisce una lista di CA considerate «universalmente» attendibili e la installa nel dispositivo. Spesso è possibile comunque modificare questa lista (in Windows detta, per esempio, *Archivio Autorità di certificazione radice attendibile*).

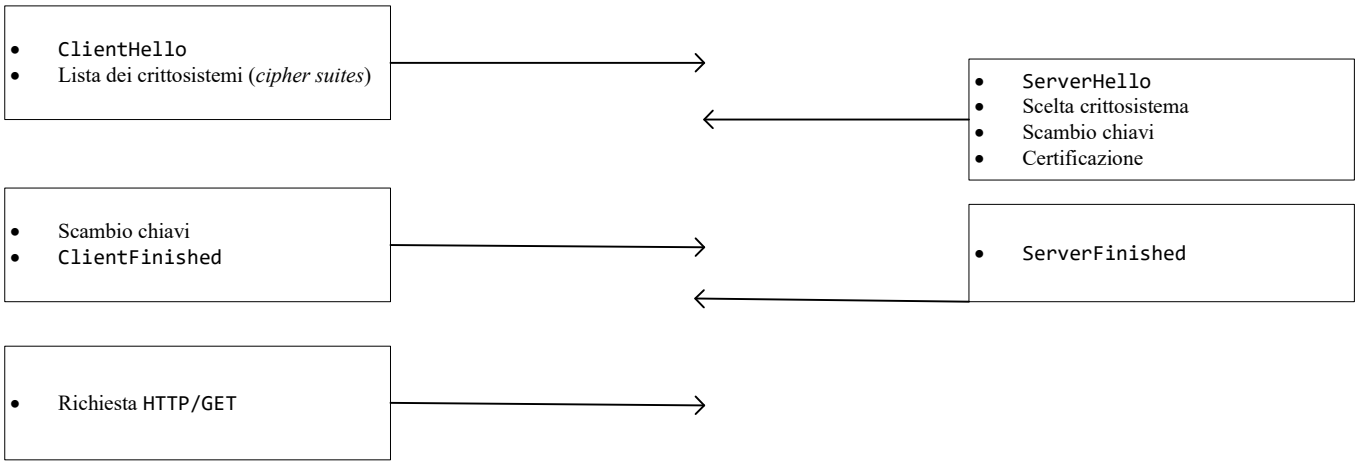

Figura 4: Schema di un handshake TLS 1.2

In questo senso è da intendersi il sottotitolo di questo Lavoro – *L'eterno compromesso tra sicurezza e usabilità* – poiché siamo costretti a scendere a compromessi (comunque più che accettabili) per poter utilizzare comodamente i nostri dispositivi informatici e telematici e sfruttare le loro potenzialità in tutta sicurezza.

Conclusione

Come il lettore avrà certamente potuto comprendere, la Crittografia è una disciplina che ha conosciuto un'evoluzione straordinaria nel corso dei millenni. Quella di oggi è una Crittografia fondata su solide basi matematiche, coadiuvata dalla fino a pochi decenni fa inimmaginabile potenza di calcolo dei computer, che deve però convivere ed adattarsi alle necessità e alle esigenze del mondo moderno, sia in termini di sicurezza informatica, sia in termini di celerità nello scambio di informazioni. Come si può evincere dal sottotitolo di questo Lavoro, la Crittografia del XXI secolo deve, più che mai, inserirsi in un contesto nel quale è di fondamentale importanza poter trasmettere dati in modo *sicuro* e contemporaneamente *rapido*, senza che l'utente finale debba fronteggiare eccessive *complicazioni*. Se per servire una pagina web all'utente in modo sicuro fosse necessario anche solo un minuto rispetto ai canonici pochi secondi di una connessione in chiaro, l'utilizzo che oggi facciamo del Web non sarebbe possibile. Non potremmo utilizzare Internet per inviare ad altri informazioni confidenziali di ogni genere.

Il problema della fattorizzazione dei numeri composti in modo rapido ed efficiente è sempre attuale. La sua soluzione non è ancora stata trovata, e proprio su questo assunto si basa la sicurezza dei moderni algoritmi crittografici, in particolare dell'RSA. L'evoluzione celerissima dell'Informatica stimola la ricerca di nuovi metodi matematici, e la richiesta di potenza di calcolo da parte della Matematica stimola il miglioramento nell'efficienza dei computer.

Confido nel fatto che gli esempi pratici forniti per mezzo del codice informatico da me realizzato, il lettore sia stato facilitato, accompagnato nella comprensione della matematica che sta dietro agli algoritmi crittografici più avanzati usati in questo nostro secolo.

APPENDICE — Esecuzione del codice

Alcuni codici sono stati scritti utilizzando il linguaggio Java; altri con il linguaggio Python, in versione 3. Per questo motivo è necessario seguire due procedure diverse. Recarsi innanzitutto su `https://github.com/ncavallini/LaM`.

Java

Esistono due possibilità:

Esecuzione online

1. Visitare un IDE online Java, per esempio quello offerto alla pagina `https://www.jdoodle.com/online-java-compiler-ide/`
2. Scegliere un listato dalla cartella `/java`
3. Copiare tutto il codice e sostituire interamente il contenuto già presente nella pagina dell'IDE online con il contenuto degli Appunti
4. Attivare l'interruttore *Interactive*
5. Cliccare sul pulsante *Execute* e attendere la compilazione.

Esecuzione in locale

1. Scaricare e installare il *Java SE SDK* dal sito ufficiale (`https://www.oracle.com/java/technologies/javase-downloads.html`)
2. Scaricare e installare un IDE Java. Si consiglia l'utilizzo di Eclipse (`https://www.eclipse.org/downloads/`)
3. Scaricare un listato di codice dalla cartella `/java` sul proprio PC ed aprirlo con Eclipse;
4. Rinominare il file così che il nome sia `Nome_ Classe.java`. È importante che il nome del file sia **identico** al nome della classe.
5. Avviare la compilazione con il tasto ➔.

Python 3

Esistono due possibilità:

Esecuzione online

1. Visitare un IDE online Python, per esempio quello offerto alla pagina `https://www.jdoodle.com/python3-programming-online/`
2. Scegliere un listato dalla cartella `/python3`;
3. Copiare tutto il codice e sostituire interamente il contenuto già presente nella pagina dell'IDE online con il contenuto degli Appunti
4. Attivare l'interruttore *Interactive*.
5. Cliccare sul pulsante *Execute* e attendere la compilazione.

Esecuzione locale

1. Scaricare Python dal sito ufficiale (`https://www.python.org/downloads/`)
2. Scegliere e scaricare un listato di codice dalla cartella `/python3`
3. Aprire, nella directory contenente il file scaricato, una *shell*:
   - Windows: **Prompt dei comandi** – Premere i tasti WINDOWS + R sulla tastiera e inserire `cmd` nella casella *Esegui*. Confermare con *OK*
   - Mac OS: **Terminale** – Aprire l'app *Terminale* dal Launchpad o dalla cartella `Applicazioni > Utility`
   - Linux: **Terminale** – Aprire il programma *Terminale* con la combinazione di tasti SUPER + T (WINDOWS + T sui PC con tastiera Windows) .
4. Digitare `python nome_file.py`
5. Confermare con INVIO.

Riferimenti bibliografici e sitografici

Bibliografia

[1] Anna Bernasconi, Paolo Ferragina e Fabrizio Luccio. *Elementi di crittografia*. Pisa: Pisa University Press, 2016.

[2] Harold Davenport. *Aritmetica superiore: un'introduzione alla teoria dei numeri*. Bologna: Zanichelli, 1994.

[3] Niels Ferguson, Bruce Schneier e Tadayoshi Kohno. *Il manuale della Crittografia: applicazioni pratiche dei protocolli crittografici*. Milano: Apogeo, 2011.

[4] Jonathan Katz e Yehuda Lindell. *Introduction to modern cryptography: principles and protocols*. Londra: Chapman & Hall, 2007.

[5] Alessandro Languasco e Alessandro Zaccagnini. *Manuale di crittografia: teoria, algoritmi e protocolli*. Milano: Hoepli, 2015.

[6] Claude E. Shannon. «Communication Theory of Secrecy Systems». In: *Bell System Technical Journal* 28.4 (1949), 656–715. DOI: 10.1002/j.1538-7305.1949.tb00928.x. URL: http://pages.cs.wisc.edu/~rist/642-spring-2014/shannon-secrecy.pdf (visitato il 14/07/2020).

Sitografia

[7] Paolo Bonavoglia. *Aritmetica finita: il generatore*. 2020. URL: http://www.crittologia.eu/mate/generatore.html (visitato il 18/08/2020).

[8] Paolo Bonavoglia. *L'algoritmo DH (Diffie-Hellman)*. 2020. URL: http://www.crittologia.eu/critto/dh.html (visitato il 11/08/2020).

[9] Paolo di Febbo. *Crittografia asimmetrica schema — Wikimedia Commons, the free media repository*. 2016. URL: https://commons.wikimedia.org/w/index.php?title=File:Crittografia_asimmetrica_schema.png&oldid=226296387 (visitato il 18/08/2020).

[10] La crittografia tra passato e futuro. *La crittografia asimmetrica*. URL: https://sites.google.com/site/lacrittografiasimmetrica/home/la-crittografia-assimetrica (visitato il 12/08/2020).

[11] La crittografia tra passato e futuro. *La crittografia simmetrica*. URL: https://sites.google.com/site/lacrittografiasimmetrica/home/la-crittografia-simmetrica (visitato il 12/08/2020).

[12] RSA Laboratories. *RSA Security*. 2004–2020. URL: http://rsasecurity.com (visitato il 01/10/2020).

[13] Università degli Studi di Milano. *Frequenza dei caratteri in italiano*. URL: http://www.mat.unimi.it/users/labpls/allegati/distribuzione_italiano.pdf (visitato il 14/07/2020).

[14] CrypTool Online. *AES (step-by-step)*. URL: https://www.cryptool.org/en/cto/highlights/aes-step-by-step.html (visitato il 27/10/2020).

[15] Claudio Picciarelli. *Crittografia asimmetrica (a chiave pubblica)*. Presentazione PowerPoint. URL: https://avires.dimi.uniud.it/claudio/teach/sicurezza2013/lezione-04.pdf (visitato il 12/08/2020).

[16] Similarweb. *Google.com – July 2020 overview*. 2020. URL: https://www.similarweb.com/website/google.com/ (visitato il 11/08/2020).

[17] Shubham Singh. *Primitive root of a number n modulo n*. 2019. URL: https://www.geeksforgeeks.org/primitive-root-of-a-prime-number-n-modulo-n/ (visitato il 29/09/2020).