

# Crittografia

Fabio Stumbo  
Dipartimento di Matematica  
Università di Ferrara  
Ferrara, I  
`f.stumbo@unife.it`

<i>INDICE</i>	2
---------------	---

## **Indice**

<b>1</b>	<b>Introduzione</b>	<b>3</b>
<b>2</b>	<b>Preliminari algebrici</b>	<b>4</b>
2.1	L'algoritmo euclideo . . . . .	5
2.2	Identità di Bézout . . . . .	7
2.3	Le congruenze . . . . .	9
2.4	Classi di congruenze . . . . .	11
2.5	Potenze “grandi” di una classe di congruenza . . . . .	12
2.6	Il Teorema di Eulero . . . . .	14
2.7	Test di primalità . . . . .	15
<b>3</b>	<b>Crittografia</b>	<b>17</b>
3.1	Cifrari simmetrici . . . . .	18
3.2	Cifrari asimmetrici . . . . .	26
	<b>Riferimenti bibliografici</b>	<b>30</b>
	<b>Indice analitico</b>	<b>31</b>

*It may well be doubted whether human ingenuity can construct an enigma of the kind which human ingenuity may not, by proper application, resolve.*

E. A. Poe, *The gold bug*

## 1 Introduzione

La crittografia è quella scienza che si occupa di trasformare un testo, un messaggio, il flusso di una comunicazione in modo che un osservatore diverso dal legittimo destinatario sia impossibilitato a recuperarne il significato. Fino a non molto tempo fa, si trattava di una disciplina sostanzialmente semiconosciuta e ignorata in ambiente accademico, considerato il fatto che la crittografia veniva usata in ambito essenzialmente militare per cui ogni nazione studiava in segreto i propri cifrari, cercando di custodire gelosamente gli algoritmi utilizzati, e con altrettanto segreto cercava di violare i codici delle altre nazioni. Questo è uno dei motivi per cui la crittografia è stata ammantata di un'aurea di ambiguità, a metà tra la scienza e la magia.

È solo negli ultimi decenni, con lo sviluppo delle reti mondiali delle comunicazioni e del commercio elettronico, che si diffonde la necessità di avere comunicazioni riservate ed inviolabili anche tra la generica popolazione civile. Gli algoritmi di crittografia iniziano quindi ad essere studiati in modo sistematico ed approfondito.

Se fino a questo punto la matematica era considerata collaterale alla crittografia, con l'avvento degli algoritmi di cifratura a chiave pubblica essa diventa uno strumento essenziale dal quale non si può prescindere, al punto che anche argomenti matematici più "esoterici" e fino ad allora con poche applicazioni pratiche come i campi finiti, gli algoritmi di fattorizzazione e le curve ellittiche, diventano di uso quotidiano per i crittografi che quindi, sostanzialmente, devono in primo luogo essere dei matematici.

In queste brevi note vedremo nella sezione 3 alcuni cifrari storici ma comunque interessanti, fino ad arrivare ad un famoso algoritmo di cifratura a chiave pubblica largamente utilizzato nella pratica quotidiana: l'algoritmo RSA. Nella sezione 2 svilupperemo gli argomenti necessari per poter comprendere appieno il funzionamento di tutti i cifrari che presenteremo. La trattazione sarà di tipo informale e non troppo rigorosa dal punto di vista matematico, in modo da poter essere fruibile, si spera, anche con conoscenze molto limitate. Osserviamo che queste nozioni sono anche sufficienti per la comprensione di altri cifrari di uso comune che però non tratteremo, come, per esempio, il cifrario ElGamal che sta alla base dell'algoritmo DSA, lo standard utilizzato per definire la firma digitale.

Non tratteremo, invece, i due cifrari simmetrici più utilizzati: DES ed

AES. Il primo perché non è analizzabile se non nella sua mera descrizione esecutiva, dato che le motivazioni che stanno alla base di alcune fasi fondamentali non sono state divulgate dalla NSA, che ha sviluppato l'algoritmo insieme alla IBM. Inoltre, è un cifrario che sta per essere sostituito con l'AES perché ormai ritenuto insicuro. Quanto a quest'ultimo, per poterlo analizzare è necessario uno studio abbastanza approfondito del campo finito  $GF(256)$ , che va al di là di quanto possiamo fare in queste note.

## 2 Preliminari algebrici

In questa sezione vediamo gli strumenti matematici necessari per poter esaminare i principali algoritmi di crittografia. Supporremo noti alcuni concetti "elementari" di teoria dei numeri, come la fattorizzazione dei numeri interi in prodotto di numeri primi e l'algoritmo della divisione euclidea tra numeri interi.

Dati due numeri  $a, b \in \mathbb{Z}$  un divisore comune dei due numeri è un numero  $d$  tale che  $d$  divida sia  $a$  che  $b$ . In formule,  $d|a$  e  $d|b$ . Detto ciò, possiamo definire il massimo comun divisore tra  $a$  e  $b$ :

DEFINIZIONE 2.1. Il massimo comune divisore di  $a$  e  $b$  è il massimo tra tutti i divisori comuni di  $a$  e  $b$ :

$$\text{MCD}(a, b) \doteq \max\{d \text{ t.c. } d|a \wedge d|b\}.$$

OSSERVAZIONE 2.2. In realtà il termine "massimo" andrebbe interpretato con riferimento all'ordinamento di divisibilità tra numeri interi ( $a \preceq b \Leftrightarrow a|b$ ) e non rispetto all'ordinamento naturale. In questo modo risulta più chiaro perché anche nel caso dei numeri interi sia più corretto dire "un" MCD invece che "il" MCD: per esempio, tanto 6 che  $-6$  sono, con questo ordinamento, più grandi di tutti gli altri divisori comuni di 18 e 30. Nel caso dei numeri interi, comunque, quando si dice "il" MCD è da intendersi quello positivo.

Per calcolare il MCD di due numeri esistono vari metodi. Essendo una proprietà che riguarda i divisori di due numeri, la strada più spontanea è quella di fattorizzare i numeri in questione: se  $a = p_1^{n_1} \cdots p_r^{n_r}$  è la fattorizzazione in fattori primi di  $a$  e  $b = q_1^{m_1} \cdots q_s^{m_s}$  è quella di  $b$ , tutti i divisori comuni si trovano facilmente considerando un qualsiasi prodotto dei numeri primi comuni alle due fattorizzazioni con un esponente che non sia superiore al minimo tra i due esponenti relativi.

Tale metodo è, per quanto naturale, estremamente poco efficiente allo stato attuale: gli algoritmi noti per determinare la fattorizzazione di un

numero intero sono molto lenti: sono tutti di tipo “*esponenziale*” o “*sub-esponenziale*”.

Esiste invece un algoritmo, tanto antico quanto efficiente, che permette di effettuare tale calcolo in “*tempo polinomiale*”. Stiamo parlando, naturalmente, dell’algoritmo euclideo.

## 2.1 L’algoritmo euclideo

L’algoritmo euclideo si basa su una proprietà fondamentale della divisione euclidea:

DEFINIZIONE 2.3. Dati due interi  $a, b$  esistono, e sono unici, due interi  $q, r$  (detti, rispettivamente, *quoziente* e *resto*) tali che

$$a = bq + r \quad \text{e} \quad 0 \leq r \leq b - 1. \quad (1)$$

Con questa notazione, abbiamo

TEOREMA 2.4 (Euclide).

$$\text{MCD}(a, b) = \text{MCD}(b, r)$$

*Dimostrazione.* Dimostreremo una proprietà più forte, da cui segue il teorema:

$$\{d \text{ t.c. } d|a \wedge d|b\} = \{d \text{ t.c. } d|b \wedge d|r\}.$$

Dovendo dimostrare un’uguaglianza tra due insiemi, dobbiamo dimostrare una doppia inclusione.

Chiamiamo

$$\begin{aligned} X &= \{d \text{ t.c. } d|a \wedge d|b\} \\ Y &= \{d \text{ t.c. } d|b \wedge d|r\}. \end{aligned}$$

“ $\subseteq$ ”: Sia  $d \in X$  un divisore comune di  $a$  e  $b$ : possiamo quindi scrivere  $a = da'$  e  $b = db'$ . Scegliendo  $q$  ed  $r$  come in 1, si ha

$$r = a - bq = da' - db'q = d(a' - b'q)$$

e quindi  $d|r$ , cioè  $d \in Y$ .

“ $\supseteq$ ”: Sia  $d \in Y$ :  $b = db'$  e  $r = dr'$ . Allora

$$a = bq + r = db'q + dr' = d(b'q + r')$$

e quindi  $d|a$  e  $d \in X$ . □

Questo teorema permette di ridurre il problema del calcolo del MCD tra  $a$  e  $b$  al problema del calcolo del MCD tra  $b$  ed  $r$ . Applicando nuovamente il teorema partendo questa volta da  $b$  ed  $r$ , si vede che si può proseguire con divisioni euclidee successive fino a che il resto non diventa nullo. Da

$$\begin{aligned}
 a &= bq_1 + r_2 \\
 b &= r_2q_2 + r_3 \\
 r_2 &= r_3q_3 + r_4 \\
 &\vdots \\
 r_{n-1} &= r_nq_n + r_{n+1} \\
 r_n &= r_{n+1}q_{n+1}
 \end{aligned} \tag{2}$$

si ricava

$$\begin{aligned}
 \text{MCD}(a, b) &= \text{MCD}(b, r_2) \\
 &= \text{MCD}(r_2, r_3) \\
 &\vdots \\
 &= \text{MCD}(r_{n-1}, r_n) \\
 &= \text{MCD}(r_n, r_{n+1}) = r_{n+1}.
 \end{aligned}$$

Quindi, per calcolare il massimo comun divisore tra  $a$  e  $b$  è sufficiente fare le divisioni ripetute 2: l'ultimo resto non nullo è il MCD cercato.

Scritto in pseudocodice, l'algoritmo diventa:

**Algoritmo 2.1:** ALGORITMO EUCLIDEO ( $m, n$ )

```

a ← m
b ← n
while b ≠ 0
  do {
    q ← ⌊a/b⌋
    r ← a - bq
    a ← b
    b ← r
  }
return (a)

```

È possibile dimostrare che quest'algoritmo ha un'efficienza polinomiale: per maggiori informazioni si veda [1].

Nell'implementazione dell'algoritmo euclideo, un'ulteriore semplificazione si ha se si osserva che la divisione euclidea tra due numeri può essere semplicemente considerata come una sottrazione del più piccolo dal più grande tante volte quante ciò è possibile. In pseudocodice questo si traduce con

**Algoritmo 2.2:** ALGORITMO EUCLIDEO  $(m, n)$

```

a ← m
b ← n
while a ≠ b
  do { if a > b
       then a ← a - b
       else b ← b - a
    }
return (a)

```

## 2.2 Identità di Bézout

L'algoritmo euclideo permette anche di ricavare per via elementare una proprietà del MCD che è di fondamentale importanza: l'identità di Bézout.

**TEOREMA 2.5.** *Con riferimento alla notazione delle divisioni ripetute 2, si ponga  $x_0 = 1$ ,  $x_1 = 0$ ,  $y_0 = 0$ ,  $y_1 = 1$  e, per induzione, si definisca*

$$\begin{cases} x_{i+2} = x_i - q_{i+1}x_{i+1} \\ y_{i+2} = y_i - q_{i+1}y_{i+1} \end{cases}$$

Con queste definizioni, si ha (posto  $r_0 = a$ ,  $r_1 = b$ )

$$r_i = x_i a + y_i b \quad \text{per } i = 0, \dots, n+1. \quad (3)$$

Dal teorema ricaviamo in particolare l'identità di Bézout: dati  $a, b$  esistono  $\lambda, \mu$  tali che

$$\text{MCD}(a, b) = \lambda a + \mu b. \quad (4)$$

*Dimostrazione.* L'equazione 3 è banalmente verificata per  $i = 0, 1$ . Dimostriamo per induzione che è vera sempre. Per  $i > 1$  si ha

$$\begin{aligned} x_i a + y_i b &= (x_{i-2} - q_{i-1}x_{i-1})a + (y_{i-2} - q_{i-1}y_{i-1}b) \\ &= (x_{i-2}a + y_{i-2}b) - q_{i-1}(x_{i-1}a + y_{i-1}b) \\ &= r_{i-2} - q_{i-1}r_{i-1} \\ &= r_i \end{aligned}$$

dove la penultima equazione segue dall'ipotesi induttiva e l'ultima segue dalla definizione di  $r_i$  in 2.  $\square$

Il teorema precedente permette di implementare in modo molto efficiente una versione estesa dell'algoritmo euclideo che calcola anche i coefficienti dell'identità di Bézout:

**Algoritmo 2.3:** ALGORITMO EUCLIDEO, VERSIONE ESTESA( $m, n$ )

```

 $a \leftarrow m$ 
 $b \leftarrow n$ 
 $s \leftarrow 1$ 
 $s_1 \leftarrow 0$ 
 $t \leftarrow 0$ 
 $t_1 \leftarrow 1$ 
while  $b \neq 0$ 
   $q \leftarrow \lfloor \frac{a}{b} \rfloor$ 
   $r \leftarrow a - bq$ 
   $a \leftarrow b$ 
   $b \leftarrow r$ 
  do  $temp \leftarrow s - qs_1$ 
      $s \leftarrow s_1$ 
      $s_1 \leftarrow temp$ 
      $temp \leftarrow t - qt_1$ 
      $t \leftarrow t_1$ 
      $t_1 \leftarrow temp$ 
return  $(a, s, t)$ 
osservazione:  $a = \text{MCD}(m, n)$  e  $a = sm + tb$ 

```

Usando la definizione e l'identità di Bézout, è possibile dimostrare molte utili proprietà del massimo comun divisore. Ne presentiamo alcune come esercizio.

ESERCIZI 2.6.

1. Se  $d|a$  e  $d|b$  allora  $d|\text{MCD}(a, b)$ .
2. Se  $\text{MCD}(a, b) = 1$  allora  $\text{MCD}(ab, c) = \text{MCD}(a, c)\text{MCD}(b, c)$ .
3. Se  $\text{MCD}(a, b) = 1$  e  $c|b$  allora  $\text{MCD}(a, c) = 1$ .
4. Se  $a|bc$  e  $\text{MCD}(a, b) = 1$  allora  $a|c$ .
5. Dato  $e$ , esistono  $m, n$  tali che  $ma + nb = e$  se, e solo se,  $\text{MCD}(a, b)|e$ .
6.  $m\text{MCD}(a, b) = \text{MCD}(ma, mb)$ .
7. Se  $d = \text{MCD}(a, b)$  allora  $\text{MCD}(a/d, b/d) = 1$ .

“Dualmente” al massimo comun divisore, è possibile definire il minimo comune multiplo di due numeri:

DEFINIZIONE 2.7. Il minimo comune multiplo di  $a$  e  $b$  è il minimo tra tutti i multipli comuni di  $a$  e  $b$  :

$$\text{mcm}(a, b) \doteq \min\{d \text{ t.c. } a|d \wedge b|d\}.$$

La relazione fondamentale che lega il massimo comun divisore al minimo comune multiplo è

$$\text{MCD}(a, b) \cdot \text{mcm}(a, b) = ab \quad (5)$$

### 2.3 Le congruenze

Le nozioni di congruenza e classe di congruenza sono due nozioni con le quali siamo abituati a confrontarci ogni giorno, sebbene inconsapevolmente. Due esempi su tutti ricorrono in particolare nella nostra quotidianità. In primo luogo, abbiamo le congruenze modulo 12: ogni volta che guardiamo un orologio (di quelli a lancette. . .) facciamo automaticamente una congruenza per stabilire in quale momento della giornata ci troviamo. Infatti ogni 12 ore le lancette riprendono la stessa posizione e quindi l'ora del giorno è riportata sull'orologio a meno di un multiplo di 12: è rappresentata solo la sua classe di resto modulo 12. Serve una nostra interpretazione per stabilire se è necessario aggiungere 12 per ottenere l'ora corretta.

Un altro esempio ormai familiare, nell'era dei computer, è quello delle congruenze modulo 2: nessuno ha più difficoltà ad interpretare l'aritmetica dei bit 0, 1, vale a dire le tabelle

$$\begin{array}{c|cc} + & 0 & 1 \\ \hline 0 & 0 & 1 \\ 1 & 1 & 0 \end{array} \quad \begin{array}{c|cc} \times & 0 & 1 \\ \hline 0 & 0 & 0 \\ 1 & 0 & 1 \end{array}$$

e anche la trasposizione di queste tabelle in termini di insiemi di numeri

$$\begin{array}{c|cc} + & \text{pari} & \text{dispari} \\ \hline \text{pari} & \text{pari} & \text{dispari} \\ \text{dispari} & \text{dispari} & \text{pari} \end{array} \quad \begin{array}{c|cc} \times & \text{pari} & \text{dispari} \\ \hline \text{pari} & \text{pari} & \text{pari} \\ \text{dispari} & \text{pari} & \text{dispari} \end{array}$$

è abbastanza spontanea.

Veniamo dunque alle definizioni matematiche di questi concetti.

DEFINIZIONE 2.8. Due interi  $a, b \in \mathbb{Z}$  si dicono congruenti modulo  $m \in \mathbb{Z}$  se  $m|a-b$ , equivalentemente, esiste  $k \in \mathbb{Z}$  tale che  $a = b + km$ . La notazione usata è

$$a \equiv b \pmod{m}.$$

Un caso particolare è che  $a \equiv 0 \pmod{m}$  se, e solo se,  $m|a$ .

PROPOSIZIONE 2.9. Ogni intero è congruente, modulo  $m$ , ad uno ed uno solo degli interi dell'insieme  $\{0, 1, 2, \dots, m-1\}$ .

*Dimostrazione.* Basta applicare la divisione euclidea 2.3: esiste un unico  $q$  ed un unico  $r$  tali che  $a = mq + r$  e  $0 \leq r < m$ .  $\square$

La divisione euclidea permette anche di dare un criterio basato sui resti per determinare se due numeri sono congrui modulo  $m$ .

TEOREMA 2.10. Siano  $a, b, m$  tre numeri interi e siano  $r, s$  i resti, rispettivamente, delle divisioni di  $a$  e  $b$  rispetto ad  $m$ ; allora

$$a \equiv b \pmod{m} \iff r = s.$$

*Dimostrazione.* Per ipotesi, esistono  $u, v$  tali che  $a = mu + r$  e  $b = mv + s$ .  
 $(\Leftarrow)$  Se  $r = s$  allora  $a - mu = b - mv$ , da cui  $a - b = m(u - v)$  e quindi  $a \equiv b \pmod{m}$ .

$(\Rightarrow)$  Viceversa, sia  $a \equiv b \pmod{m}$ , cioè  $a - b = km$  per qualche  $k$ ; allora

$$\begin{aligned} km &= a - b \\ &= mu + r - (mv + s) \\ &= m(u - v) + r - s \end{aligned}$$

da cui

$$r - s = m(k + v - u).$$

D'altra parte,  $0 \leq r, s < m$  e quindi  $-(m-1) \leq -s < 0$ , da cui  $-(m-1) \leq r - s < m-1$ . Ciò vuol dire che  $r - s$  deve essere un multiplo di  $m$  appartenente all'intervallo  $[-(m-1), m-1]$  ma, in tale intervallo, l'unico multiplo di  $m$  è lo 0, quindi  $r - s = 0$  e  $r = s$ .  $\square$

Lasciamo come esercizio alcune proprietà di base delle congruenze.

ESERCIZI 2.11. Siano  $a, b, c, a', b', k$  ed  $n$  degli interi. Si ha:

1. se  $a \equiv b \pmod{n}$  allora  $ka \equiv kb \pmod{n}$ ;
2. se  $a \equiv b \pmod{n}$  e  $b \equiv c \pmod{n}$  allora  $a \equiv c \pmod{n}$ ;
3. se  $a \equiv b \pmod{n}$  e  $a' \equiv b' \pmod{n}$  allora  $a + a' \equiv b + b' \pmod{n}$  e  $aa' \equiv bb' \pmod{n}$ .

OSSERVAZIONE 2.12. Fra tutte le proprietà, una che *non* vale è quella di cancellazione: non è vero che se  $ab \equiv ac \pmod{n}$  allora  $b \equiv c \pmod{n}$ . Come esempio, si consideri  $2 \cdot 1 \equiv 2 \cdot 3 \pmod{4}$  ma  $1 \not\equiv 3 \pmod{4}$ . Questa legge vale solo nel caso in cui  $\text{MCD}(a, n) = 1$ .

*Dimostrazione.* Sia  $ab \equiv ac \pmod{n}$  e  $\text{MCD}(a, n) = 1$ . Allora  $n \mid ab - ac = a(b - c)$ . Dal fatto che  $\text{MCD}(a, n) = 1$  segue che  $n \mid b - c$ , cioè  $b \equiv c \pmod{n}$ .  $\square$

Più in generale,

PROPOSIZIONE 2.13. *Se  $ab \equiv ac \pmod{n}$  allora*

$$b \equiv c \left( \text{mod} \frac{n}{\text{MCD}(n, a)} \right).$$

*Dimostrazione.* Sia  $d = \text{MCD}(a, n)$  e si scriva  $a = a'd, n = n'd$ . Si ha che  $\text{MCD}(a', n) = 1 = \text{MCD}(a', n')$ .

Da  $ab \equiv ac \pmod{n}$  segue  $a(b - c) = kn$  per qualche  $k$ . Sostituendo,  $a'd(b - c) = kn'd$ , per cui  $a'(b - c) = kn'$ , che è equivalente alla tesi.  $\square$

## 2.4 Classi di congruenze

Dal punto di vista matematico, la congruenza modulo un numero  $n$  è una relazione di equivalenza. Possiamo quindi ripartire  $\mathbb{Z}$  in classi di equivalenza,

DEFINIZIONE 2.14. Dati due interi  $a, n$ , la classe di congruenza (o anche classe di resto) di  $a$  modulo  $n$  è l'insieme

$$[a]_n = \{b \text{ t.c. } b \equiv a \pmod{n}\} = \{a + kn \text{ t.c. } k \in \mathbb{Z}\}.$$

Con questa notazione, l'esempio dei numeri binari si può anche riassumere con

$$\begin{aligned} \text{pari} &= [0]_2 \\ \text{dispari} &= [1]_2 \end{aligned}$$

e si ha che  $\mathbb{Z} = [0]_2 \cup [1]_2$ .

Più in generale, dalla proposizione 2.9 segue che, fissato un intero (positivo)  $n$ ,

$$\mathbb{Z} = [0]_n \cup [1]_n \cup \dots \cup [n-1]_n.$$

L'insieme formato dalle classi di congruenza modulo  $n$  viene indicato con il simbolo  $\mathbb{Z}/n\mathbb{Z}$  o anche, a volte (seppur impropriamente), con  $\mathbb{Z}_n$ .

In  $\mathbb{Z}/n\mathbb{Z}$  è possibile sommare e moltiplicare le classi di resto nel seguente modo:

$$\begin{aligned} [a]_n + [b]_n &= [a + b]_n \\ -[a]_n &= [-a]_n \\ [a]_n \cdot [b]_n &= [ab]_n. \end{aligned}$$

OSSERVAZIONE 2.15. La precedente definizione, in realtà è più complessa di quanto possa sembrare a prima vista. Infatti noi trattiamo come dei singoli oggetti le classi di congruenza, che invece sono degli insiemi.

Il problema che nasce è il seguente. La stessa classe di congruenza può essere rappresentata da elementi diversi:  $[a]_n = [\alpha]_n$  se  $n|(a - \alpha)$ . Analogamente,  $[b]_n = [\beta]_n$  se  $n|(b - \beta)$ . Ma allora per la somma si avrebbero due definizioni distinte: da una parte  $[a]_n + [b]_n = [a + b]_n$  e dall'altra  $[a]_n + [b]_n = [\alpha]_n + [\beta]_n = [\alpha + \beta]_n$ .

Sarebbe quindi necessario verificare che  $[a + b]_n = [\alpha + \beta]_n$  per essere sicuri di aver dato una buona definizione.

Con queste definizioni di somma e prodotto, l'insieme  $\mathbb{Z}/n\mathbb{Z}$  verifica tutte le usuali proprietà: associativa, commutativa, elemento neutro, distributiva, ecc. L'unica proprietà che non è in generale verificata è l'esistenza dell'inverso moltiplicativo per un elemento non nullo. Infatti:

PROPOSIZIONE 2.16. *Data una classe di resto  $[a]_n$  esiste una classe di resto  $[b]_n$  tale che  $[a]_n \cdot [b]_n = [1]_n$  se, e solo se,  $\text{MCD}(a, n) = 1$ .*

*Dimostrazione.* Si ha che

$$\begin{aligned} [a]_n \cdot [b]_n = [1]_n &\iff \exists k \in \mathbb{Z} \text{ t.c. } ab = 1 + kn \\ &\iff \exists k \in \mathbb{Z} \text{ t.c. } ab - kn = 1 \\ &\iff \text{MCD}(a, n) = 1 \end{aligned}$$

per le proprietà derivanti dall'identità di Bézout. □

## 2.5 Potenze “grandi” di una classe di congruenza

Come vedremo, più avanti avremo bisogno di calcolare delle potenze molto alte di un numero modulo un altro numero: dovremo calcolare  $a^b \pmod{n}$  dove  $a, b$  ed  $n$  sono tutti numeri con alcune centinaia di cifre. Detto così, sembra un compito proibitivo ma in realtà se il calcolo lo facciamo direttamente in  $\mathbb{Z}_n$  diventa molto semplice ed anche di veloce esecuzione, grazie al seguente algoritmo. Si scriva  $b$  in base 2:

$$b = b_i 2^i + b_{i-1} 2^{i-1} + \dots + b_2 2^2 + b_1 2 + b_0$$

con  $b_j \in \{0, 1\}$ . Fatto ciò, si calcoli per induzione  $a^{2^{j+1}} \equiv (a^{2^j})^2 \pmod{n}$ . Adesso non resta da far altro che moltiplicare tra loro tutti quegli  $a^j$  tali che i corrispondenti  $b_j$  sono 1:

$$\begin{aligned} a^b &\equiv (a^{b_i 2^i + b_{i-1} 2^{i-1} + \dots + b_2 2^2 + b_1 2 + b_0}) \pmod{n} \\ &\equiv (a^{2^i})^{b_i} \cdot (a^{2^{i-1}})^{b_{i-1}} \dots (a^{2^1})^{b_1} \cdot a^{b_0} \pmod{n}. \end{aligned}$$

Vediamo, per esempio, come calcolare  $12^{100} \pmod{34}$ . Innanzi tutto,  $100 = 64 + 32 + 4 = 2^6 + 2^5 + 2^2$ . Ora calcoliamo le potenze necessarie di 12:

$$\begin{aligned} 12 & \\ 12^2 &\equiv 8 \pmod{34} \\ 12^{2^2} &\equiv 8^2 \equiv 30 \pmod{34} \\ 12^{2^3} &\equiv 30^2 \equiv 16 \pmod{34} \\ 12^{2^4} &\equiv 16^2 \equiv 18 \pmod{34} \\ 12^{2^5} &\equiv 18^2 \equiv 18 \pmod{34} \\ 12^{2^6} &\equiv 18^2 \equiv 18 \pmod{34}. \end{aligned}$$

Si ha quindi

$$12^{100} = 12^{64+32+4} = 12^{2^6} 12^{2^5} 12^{2^2} \equiv 18 \cdot 18 \cdot 30 \equiv 4 \pmod{34}.$$

Questo è lo pseudocodice necessario per implementare il precedente algoritmo.

**Algoritmo 2.4:** SQUARE AND MULTIPLY( $a, b, n$ )

**osservazione:** Calcola  $a^b \pmod{n}$

$r \leftarrow 1$

$\alpha \leftarrow a \pmod{n}$

$q \leftarrow b$

**while**  $q > 1$

**do**  $\begin{cases} \text{if } q \equiv 1 \pmod{2} \\ \quad \text{then } r \leftarrow r\alpha \pmod{n} \\ \alpha \leftarrow \alpha^2 \pmod{n} \\ q \leftarrow \lfloor \frac{q}{2} \rfloor \end{cases}$

$r \leftarrow r\alpha \pmod{n}$

**return** ( $r$ )

## 2.6 Il Teorema di Eulero

Il teorema di Eulero, che è un caso particolare di un teorema più generale della teoria dei gruppi, ha un ruolo centrale nello sviluppo della crittografia a chiave pubblica. Per poterlo enunciare, abbiamo prima bisogno di definire la funzione  $\varphi$  di Eulero.

DEFINIZIONE 2.17. Dato un intero positivo  $n$ ,  $\varphi(n)$  è definita da

$$\varphi(n) = \#\{a \text{ t.c. } 1 \leq a \leq n-1 \text{ e } \text{MCD}(a, n) = 1\}.$$

$\varphi(n)$  è quindi il numero degli interi positivi minori di  $n$  che sono primi con  $n$  o anche, grazie a 2.16, il numero di classi di resto in  $\mathbb{Z}_n$  che risultano avere un inverso moltiplicativo.

Si può dimostrare che se  $n = p_1^{m_1} \cdots p_r^{m_r}$  è la fattorizzazione in numeri primi di  $n$ , allora

$$\varphi(n) = \prod_{i=1}^r p_i^{m_i-1} (p_i - 1) \cdots p_r^{m_r-1} (p_r - 1).$$

TEOREMA 2.18 (Eulero). *Se  $\text{MCD}(a, n) = 1$  allora*

$$a^{\varphi(n)} \equiv 1 \pmod{n}.$$

*Dimostrazione.* Si indichi con  $\mathbb{Z}_n^*$  l'insieme delle classi di resto relativamente prime con  $n$  e si consideri l'applicazione

$$\begin{array}{ccc} \mathbb{Z}_n^* & \longrightarrow & \mathbb{Z}_n^* \\ [b]_n & \longmapsto & [ab]_n \end{array}$$

Dato che  $\text{MCD}(a, n) = 1$ , dalle proprietà del massimo comun divisore segue che  $[b]_n \neq [c]_n \Rightarrow [ab]_n \neq [ac]_n$ . L'applicazione è dunque iniettiva e, di conseguenza, anche suriettiva. Ne segue che

$$\begin{aligned} \prod_{[b]_n \in \mathbb{Z}_n^*} [b]_n &\equiv \prod_{[b]_n \in \mathbb{Z}_n^*} [ab]_n \pmod{n} \\ &\equiv \prod_{[b]_n \in \mathbb{Z}_n^*} [a]_n [b]_n \pmod{n} \\ &\equiv [a]_n^{\#\mathbb{Z}_n^*} \prod_{[b]_n \in \mathbb{Z}_n^*} [b]_n \pmod{n} \\ &\equiv [a]_n^{\varphi(n)} \prod_{[b]_n \in \mathbb{Z}_n^*} [b]_n \pmod{n}. \end{aligned}$$

D'altra parte,  $\prod_{[b]_n \in \mathbb{Z}_n^*} [b]_n$  è ancora una classe di resto  $[c]_n$  e quindi l'equazione precedente si riduce a

$$[a]_n^{\varphi(n)} [c]_n \equiv [c]_n \pmod{n}$$

e, cancellando  $[c]_n$  grazie a 2.13, si ottiene la tesi.  $\square$

Osserviamo che nel caso particolare  $n = p$ , dove  $p$  è un numero primo, si ha che  $\varphi(p) = p - 1$  e quindi il teorema di Eulero si riduce al

TEOREMA 2.19 (Fermat). *Se  $p$  è un numero primo, allora*

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

## 2.7 Test di primalità

In quasi tutti i cifrari a chiave pubblica, compreso l'algoritmo RSA, è necessario determinare dei numeri primi molto grandi: è quindi fondamentale, affinché questi cifrari siano effettivi, disporre di un algoritmo che permetta di fare un test di primalità e che sia ragionevolmente efficiente.

Tra i vari test di primalità, uno dei più noti ed utilizzati è il test di Miller-Rabin. Lo pseudocodice relativo a quest'algoritmo è il seguente:

**Algoritmo 2.5:** MILLER-RABIN( $n$ )

```

scrivere  $n - 1 = 2^k m$  con  $m$  dispari
scegliere a caso un intero  $a$  compreso tra 1 ed  $n - 1$ 
 $b \leftarrow a^m \pmod{m}$ 
if  $b \equiv 1 \pmod{n}$ 
  then return (" $n$  è primo")
for  $i \leftarrow 0$  to  $k - 1$ 
  do  $\begin{cases} \text{if } b \equiv -1 \pmod{n} \\ \text{then return (" $n$  è primo")} \\ \text{else } b \leftarrow b^2 \pmod{n} \end{cases}$ 
return (" $n$  è composto")

```

Questo test è un test di tipo probabilistico: se il risultato del test " $n$  è primo" non abbiamo la certezza che  $n$  sia effettivamente primo, mentre se il risultato è " $n$  è composto" allora  $n$  è sicuramente composto, come segue dal

TEOREMA 2.20 (Miller-Rabin). *Se  $n$  è un numero primo, allora l'algoritmo di Miller-Rabin non può avere come uscita " $n$  è composto".*

*Dimostrazione.* Supponiamo, per assurdo, che  $n$  sia un numero primo e che l'algoritmo abbia come uscita “ $n$  è composto”.

Sia  $n - 1 = 2^k m$  con  $m$  dispari e sia  $1 \leq a \leq n - 1$  un numero scelto a caso. Dato che la risposta dell'algoritmo è “ $n$  è composto”, si deve avere  $a^m \not\equiv 1 \pmod{n}$ . Consideriamo adesso la successione dei valori  $b$  testati nell'algoritmo. Dato che ad ogni iterazione del ciclo **for**  $b$  viene elevato al quadrato, ne segue che stiamo testando  $a^m, a^{2m}, \dots, a^{2^{k-1}m}$ . Dato che la risposta dell'algoritmo è “ $n$  è composto”, ne concludiamo

$$a^{2^i m} \not\equiv -1 \pmod{n}$$

per  $0 \leq i \leq k - 1$ .

Per ipotesi  $n$  è primo, per cui dal teorema di Fermat 2.19 segue che

$$a^{2^k m} \equiv 1 \pmod{n},$$

dato che  $n - 1 = 2^k m$ ; allora  $a^{2^{k-1}m}$  è una radice quadrata di 1 modulo  $n$ . Visto che  $n$  è primo, ci sono solo due tali radici quadrate, cioè  $\pm 1 \pmod{n}$ . Sappiamo che

$$a^{2^{k-1}m} \not\equiv -1 \pmod{n},$$

da cui segue che

$$a^{2^{k-1}m} \equiv 1 \pmod{n}.$$

Allora  $a^{2^{k-2}m}$  è una radice quadrata di 1 modulo  $n$ . Con lo stesso ragionamento,

$$a^{2^{k-2}m} \equiv 1 \pmod{n}.$$

Ripetendo ciò, alla fine otteniamo

$$a^m \equiv 1 \pmod{n},$$

che è una contraddizione, perché in questo caso l'algoritmo avrebbe fornito la risposta “ $n$  è primo”.  $\square$

Ma qual è il valore di un test del quale non sia affidabile il risultato “ $n$  è primo”? Non lo dimostreremo, ma la seconda parte (di ben più complicata dimostrazione) del Teorema di Miller–Rabin afferma che la probabilità che questa risposta sia sbagliata è di  $1/4$ . Pertanto, se si applica ripetutamente, diciamo  $m$  volte, l'algoritmo allo stesso numero  $n$ , la probabilità che tutte ed  $m$  le volte si abbia il risultato “ $n$  è primo” quando invece  $n$  è composto è di  $(1/4)^m$  e quindi tale probabilità può essere resa piccola a piacere.

Il motivo per cui si usa nelle applicazioni un test probabilistico invece di uno deterministico è legato al fatto che non sono noti test di primalità deterministici molto efficienti. Di fatto, i migliori test deterministici noti

hanno un'efficienza di tipo subesponenziale, mentre il test di Miller–Rabin ha un'efficienza di tipo polinomiale. Detto in termini un po' più semplicistici, un algoritmo di tipo non polinomiale ha un tempo di esecuzione enorme su numeri con alcune centinaia di cifre mentre un algoritmo di tipo polinomiale viene eseguito in pochi secondi, sugli stessi numeri. Beninteso, stiamo parlando di esecuzione su di un calcolatore...

### 3 Crittografia

In questa sezione, vedremo alcuni algoritmi di crittografia. Alcuni sono solo di interesse storico; per contro, l'algoritmo RSA è quanto mai attuale: esso è infatti uno dei principali algoritmi a chiave pubblica attualmente in uso.

Un algoritmo di crittografia, o cifrario, è una funzione che ha due variabili in ingresso: un messaggio ed una parola chiave. La parola chiave è usata per cifrare, o decifrare, il messaggio. I cifrari si suddividono principalmente in due grandi famiglie: i cifrari simmetrici e quelli asimmetrici. I cifrari simmetrici sono quelli in cui la chiave necessaria per decifrare un messaggio coincide con la chiave che è stata usata per cifrarlo, oppure è facilmente ricavabile da essa; in quelli asimmetrici, invece, si usano due chiavi distinte: una per cifrare ed una per decifrare. Le due chiavi, in genere, sono legate tra loro da qualche proprietà matematica però non deve essere possibile ricavare una chiave dall'altra, se non teoricamente almeno praticamente.

Tutti i cifrari usati storicamente fino agli inizi degli anni settanta appartengono alla famiglia dei cifrari simmetrici: i cifrari asimmetrici e, con essi, la crittografia a chiave pubblica sono stati teorizzati nel 1976 da Diffie ed Hellman e, un anno dopo, è stato inventato il cifrario RSA, dalle iniziali dei suoi autori Rivest, Shamir ed Adleman. Altri cifrari a chiave pubblica sono stati sviluppati in seguito, tra cui citiamo il cifrario ElGamal, utilizzato insieme alla funzione di hash SHA-1 per definire lo schema di firma digitale DSA.

Nello sviluppare i cifrari, indicheremo con  $\mathcal{P}$  lo spazio dei messaggi in chiaro, con  $\mathcal{C}$  lo spazio dei messaggi cifrati e con  $\mathcal{K}$  lo spazio delle chiavi. Infine, con  $c_K(x)$  indicheremo l'operazione di cifratura di un messaggio  $x$  usando la chiave  $K \in \mathcal{K}$ , mentre con  $d_K(y)$  indicheremo l'operazione inversa di decifratura. In prima battuta si può pensare che lo spazio dei messaggi in chiaro coincida con l'insieme di tutti i messaggi di qualsiasi lunghezza aventi senso compiuto in una determinata lingua; invece, in genere un algoritmo non opera sul messaggio nella sua interezza ma suddivide il messaggio in blocchi di una lunghezza fissata e poi opera singolarmente su ciascun blocco.  $\mathcal{P}$  è allora formato dall'insieme di tali blocchi. In genere si ha che  $\mathcal{P} = \mathcal{C}$ .

Si potrebbe essere indotti a pensare che custodire gelosamente l'algoritmo usato per definire un cifrario sia una ulteriore garanzia di sicurezza per

il cifrario stesso. Così, in effetti, non è, per molti motivi. Innanzi tutto, se un algoritmo è “segreto”, esso può non venire analizzato in modo sufficientemente approfondito in modo da evidenziarne le eventuali falle. In secondo luogo, se l'algoritmo dovesse essere scoperto dai nostri “nemici”, esso potrebbe essere analizzato in modo da rendere vulnerabili i nostri messaggi. Ciò, per esempio, è quanto è successo nel corso della seconda guerra mondiale con il cifrario Enigma usato dai tedeschi: l'algoritmo, in sé, era sostanzialmente inattaccabile. L'implementazione fatta dai tedeschi, invece, indeboliva di parecchio la forza crittografica del cifrario. Nel momento in cui gli alleati sono riusciti ad entrare in possesso di una delle macchine Enigma usate dai tedeschi per cifrare i messaggi, concentrando i loro sforzi sulle falle dell'implementazione gli alleati sono riusciti a forzare il cifrario e quindi erano in grado di intercettare tutte le comunicazioni fra il quartier generale e le varie unità operative.

Infine, un ultimo fondamentale motivo per cui l'algoritmo si deve supporre che l'algoritmo sia pubblico è dato dal fatto che altrimenti non sarebbe possibile implementarlo in software distribuibili ad utenti generici e quindi non sarebbe possibile nessun tipo di transazione elettronica protetta, impedendo così qualsiasi forma di commercio elettronico.

Per tutti tali motivi, risulta chiaro perché alla base della crittografia moderna ci sia il

PRINCIPIO DI KERCKHOFFS: *la sicurezza di un cifrario deve risiedere tutta nella chiave.*

### 3.1 Cifrari simmetrici

I cifrari simmetrici che vedremo sono di interesse essenzialmente storico: non tratteremo invece cifrari attuali come il DES (Data Encryption Standard) o il Rijndael (il cifrario usato nella definizione dell'AES, l'Advanced Encryption Standard). Lo studio del primo non richiederebbe nozioni che vanno al di là di quanto visto, però la sua analisi è alquanto noiosa; inoltre, è un cifrario che ormai, seppur ancora usato, è considerato insicuro e l'AES sta per prenderne il posto. Relativamente a quest'ultimo, il suo studio richiede invece argomenti di matematica che trascendono quanto è stato possibile esporre in queste note.

Uno dei primi cifrari di cui si ha notizia è il cosiddetto cifrario di Giulio Cesare. Esaminato con le conoscenze attuali, è un cifrario estremamente elementare: esso si basa su una semplice sostituzione delle lettere del messaggio in chiaro secondo lo schema

$$\begin{array}{ccccccccc}
 A & B & C & \dots & V & X & Y & W & Z \\
 \downarrow & \downarrow & \downarrow & & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\
 D & E & F & \dots & W & Z & A & B & C
 \end{array} \tag{6}$$

Oltre alla semplicità, l'algoritmo soffre di un difetto fondamentale: esso contravviene al Principio di Kerckhoffs, dato che la chiave è del tutto assente e la segretezza del messaggio risiede interamente nella segretezza dell'algoritmo. Nonostante ciò, questo cifrario è stato ampiamente usato dai romani, e questo non deve sorprendere: all'epoca, infatti, c'era necessità di cifrare pochi messaggi. Inoltre, un messaggio cifrato veniva inviato per "corriere espresso": un messo a cavallo partiva dal quartier generale fino ad arrivare al destinatario. Per intercettare il messaggio, bisognava materialmente strapparne di mano al messaggero... Infine, data la scarsa scolarizzazione dell'epoca, è del tutto probabile che chiunque si fosse trovato in mano un messaggio cifrato avrebbe semplicemente ammesso la sua incapacità di leggerlo, come il povero narratore de *Lo scarabeo d'oro*.

È facile, comunque, generalizzare questo cifrario: al posto della permutazione  $\sigma$ , si può usare una qualsiasi delle  $26!$  permutazioni delle lettere dell'alfabeto:

$$\begin{array}{cccccccc} A & B & C & \dots & V & X & Y & W & Z \\ \downarrow & \downarrow & \downarrow & & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ \pi(A) & \pi(B) & \pi(C) & \dots & \pi(V) & \pi(X) & \pi(Y) & \pi(W) & \pi(Z) \end{array} \quad (7)$$

dove  $\pi$  è una permutazione delle lettere. In questo modo, con la notazione descritta, si ha che  $\mathcal{P} = \mathcal{C} = \{A, B, \dots, W, Z\}$  e  $\mathcal{K} = \mathcal{S}_{26}$ , il gruppo simmetrico su 26 elementi.

La chiave necessaria per criptare il messaggio è la permutazione  $\pi$ , da cui si ricava facilmente la sua inversa necessaria per la decriptazione.

Considerando un attacco a forza bruta (un attacco, cioè, in cui si provano successivamente tutte le possibili chiavi una dopo l'altra), questo è un buon cifrario: infatti ci sono  $26! = 403291461126605635584000000 \cong 4 \times 10^{26}$  possibili permutazioni. Provarle tutte trascende ogni capacità umana ed anche con un calcolatore non è cosa agevole.

Un cifrario di questo tipo si chiama *cifrario a sostituzione monoalfabetica*, perché a lettera uguale sostituisce lettera uguale. Lo pseudocodice necessario per implementare un cifrario di questo tipo è il seguente:

**Cifrario 3.1:** CIFRARIO A SOSTITUZIONE MONOALFABETICA

Sia  $\mathcal{P} = \mathcal{C} = \mathbb{Z}_{26}$  e sia  $\mathcal{K} = \mathcal{S}_{26}$  il gruppo simmetrico su 26 elementi. Per una chiave  $\pi$  (cioè, una permutazione), si definisce

$$\begin{aligned} c_K(x) &= \pi(x) \\ d_K(y) &= \pi^{-1}(y), \end{aligned}$$

dove  $\pi^{-1}$  è la permutazione inversa di  $\pi$ .

Risulta essere un cifrario di questo tipo anche quello usato da Capitan Kidd ne *Lo scarabeo d'oro*. Esattamente come avviene nel racconto, questa classe di algoritmi cade facilmente sotto un attacco di tipo statistico. Per ovviare a ciò, è stata sviluppata un'altra categoria di cifrari, i *cifrari a sostituzione polialfabetica*. In questi cifrari si opera su ogni singola lettera del messaggio con una permutazione ma la permutazione varia ad ogni lettera. Lo pseudocodice relativo è:

**Cifrario 3.2:** CIFRARIO A SOSTITUZIONE POLIALFABETICA

Sia  $\mathcal{P} = \mathcal{C} = (\mathbb{Z}_{26})^m$ , dove  $m \geq 2$  è un intero, e sia  $\mathcal{K} = (\mathcal{S}_{26})^m$ .

Per una chiave  $\pi = (\pi_1, \dots, \pi_m)$  si definisce

$$c_K(x_1, \dots, x_m) = (\pi_1(x_1), \dots, \pi_m(x_m))$$

$$d_K(y_1, \dots, y_m) = (\pi_1^{-1}(y_1), \dots, \pi_m^{-1}(y_m)),$$

dove  $\pi_i^{-1}$  è la permutazione inversa di  $\pi_i$ .

È quindi necessario definire un intero  $m$  ed altrettante permutazioni, che vengono usate una volta in successione. Dopo averle applicate tutte si riparte dall'inizio in modo ciclico. Un esempio di cifrario di questo tipo è quello generalmente attribuito a Vigenère. Esso richiede una parola chiave e l'uso della seguente tabella:

	A	B	C	...	W	Z
A	A	B	C	...	W	Z
B	B	C	D	...	Z	A
C	C	D	E	...	A	B
⋮	⋮	⋮	⋮	...	⋮	⋮
W	W	Z	A	...	X	Y
Z	Z	A	B	...	Y	W

Per cifrare un testo con una parola chiave, si ripete la parola tante volte quante è necessario per avere una lunghezza pari alla lunghezza del testo e poi si cifra ogni lettera del messaggio con la permutazione corrispondente alla colonna indicata dalla relativa lettera della parola chiave. Per esempio, per criptare "CIFRARI" con la parola chiave "roma", si procede così :

r	o	m	a	r	o	m
C	I	F	R	A	R	I
↓	↓	↓	↓	↓	↓	↓
T	W	R	R	R	F	U

Per la prima lettera, la “C”, si considera la colonna iniziante per “r” e, in quella colonna, si individua la lettera nella quale viene trasformata la “C”, semplicemente andando a fare l’intersezione con la riga iniziante per “C”, e così via per le altre lettere.

L’esempio è oltremodo banale e semplificato, ma rende comunque bene il comportamento: a lettera uguale generalmente corrisponde lettera diversa e la stessa lettera può essere immagine di lettere diverse.

Questo esempio, in realtà, soffre del fatto che la scelta delle permutazioni distinte è molto limitata e del fatto che se la parola chiave non è molto lunga ci sono alcune tecniche crittoanalitiche per determinare la lunghezza della parola chiave e, fatto questo, il problema può essere ridotto ad uno simile a quello che si ha con una cifratura monoalfabetica.

Un altro esempio di cifratura polialfabetica è il famigerato “Enigma” usato dai tedeschi durante la seconda guerra mondiale. In quel caso si usavano dei rotori (generalmente 3 o 4) messi in serie. Ogni rotore era una permutazione e ad ogni cifratura di una lettera il primo rotore faceva una rotazione di un posto. Dopo un giro completo del primo rotore, il secondo rotore faceva una rotazione di un posto, e così via, con un meccanismo simile a quello che si ha in un classico odometro meccanico di un’automobile. In questo modo la chiave era data dalla scelta e sistemazione iniziale dei rotori, mentre il periodo dopo il quale si ripeteva lo stesso schema di cifratura era molto alto: circa  $26^3$ . Con tale periodo, si potevano spedire in tutta sicurezza messaggi di alcune centinaia di parole. Lo spazio delle chiavi risultava essere, in teoria, enorme. Ciò che compromise il cifrario fu la sua implementazione: in realtà venne usato un numero molto limitato di rotori (generalmente 7) tra i quali scegliere i 3 da usare. Ciò riduceva enormemente lo spazio delle chiavi possibili. Il motivo di questa scelta è di natura pratica: la macchina Enigma era delle dimensioni di una macchina scrivere ed ogni rotore era un disco di metallo. Il tutto doveva essere incluso in una valigetta che fosse trasportabile a mano da una persona.

Se invece è ben implementato, un cifrario polialfabetico può risultare molto robusto tutt’oggi.

I cifrari che abbiamo visto fino ad ora sono tutti cifrari “a flusso”, vale a dire agiscono su un flusso di dati e operano singolarmente su ogni informazione (nei nostri casi, su ogni singola lettera) del messaggio. La maggior parte dei cifrari attuali, in realtà, agisce su blocchi del messaggio: si prende una porzione del messaggio di dimensione fissata (un blocco, per l’appunto) e lo si modifica usando la chiave e tutte le informazioni del blocco, fino ad ottenere un nuovo blocco in cui ogni singola informazione dipende da tutti i dati del blocco in ingresso. Un esempio di cifrario a blocchi è il cifrario di Hill. Esso ha anche l’importanza storica di essere stato il primo cifrario basato interamente su operazioni matematiche: i cifrari visti finora usavano solo sostituzioni e trasposizioni delle lettere.

Per poter illustrare il cifrario di Hill è necessario usare la notazione delle matrici. Senza entrare in dettagli di algebra lineare, per noi una matrice  $2 \times 2$  sarà solo un insieme di quattro numeri

$$M = \begin{pmatrix} a & b \\ c & d \end{pmatrix}.$$

Se prendiamo un'altra matrice

$$N = \begin{pmatrix} \alpha & \beta \\ \gamma & \delta \end{pmatrix},$$

allora possiamo definire la somma ed il prodotto di due matrici tramite le formule

$$M + N = \begin{pmatrix} a + \alpha & b + \beta \\ c + \gamma & d + \delta \end{pmatrix} \quad \text{e} \quad M \cdot N = \begin{pmatrix} a\alpha + b\gamma & a\beta + b\delta \\ c\alpha + d\gamma & c\beta + d\delta \end{pmatrix}$$

Si verifica facilmente che se  $D$  è tale che  $D(ad - bc) = 1$ , allora posto

$$N = \begin{pmatrix} Dd & -Db \\ -Dc & Da \end{pmatrix}, \quad (8)$$

$N$  è una matrice tale che

$$M \cdot N = N \cdot M = I, \quad \text{dove} \quad I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

Infine, se  $v = \begin{pmatrix} v_1 \\ v_2 \end{pmatrix}$  è un vettore, si può definire il nuovo vettore ottenuto dalla moltiplicazione per  $M$  tramite

$$M \cdot v = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \cdot \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} av_1 + bv_2 \\ cv_1 + dv_2 \end{pmatrix}$$

Con questa notazione, si può definire il cifrario di Hill. Dato il messaggio in chiaro  $x$ , ogni lettera viene trasformata in un numero tra 0 e 25 (quindi in un elemento di  $\mathbb{Z}_{26}$ ) tramite lo schema

$A$	$B$	$C$	$\dots$	$V$	$X$	$Y$	$W$	$Z$
$\downarrow$	$\downarrow$	$\downarrow$	$\dots$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$
0	1	2	$\dots$	21	22	23	24	25

Il messaggio viene suddiviso ora in blocchi di due numeri. Sia  $v = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$  uno di questi blocchi. Si scelga una matrice  $2 \times 2$

$$M = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \quad \text{tale che} \quad \text{MCD}(ad - bc, 26) = 1.$$

La matrice  $M$  è la chiave del cifrario e la sua inversa  $N$  si calcola facilmente grazie all'equazione 8. Eseguendo la moltiplicazione  $M \cdot v$  si ottiene il messaggio cifrato. Le operazioni di somma e di prodotto fra numeri sono, ovviamente, da intendersi modulo 26.

Per decrittare il messaggio, è sufficiente moltiplicare per la matrice inversa  $N$ : se  $w = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = M \cdot v$ , allora

$$N \cdot w = N \cdot (M \cdot v) = (N \cdot M) \cdot v = I \cdot v = v.$$

Come esempio, proviamo l'algoritmo usando come messaggio la parola "CIFRARI", e come chiave la matrice

$$M = \begin{pmatrix} 2 & 5 \\ 1 & 3 \end{pmatrix}$$

la cui inversa è

$$N = \begin{pmatrix} 3 & 21 \\ 25 & 2 \end{pmatrix}$$

(si ricordi che anche i coefficienti della matrice sono da intendersi modulo 26).

Si può osservare che la parola da codificare ha lunghezza dispari, per cui non è possibile suddividerla in blocchi di lunghezza 2. Per ovviare a ciò, si usa la cosiddetta tecnica del *padding*, cioè si aggiungono dei caratteri arbitrari fino ad avere la lunghezza desiderata. Per esempio, aggiungiamo la lettera "X". Dobbiamo quindi codificare "CIFRARIX". Tradotta in numeri, otteniamo la successione  $\{2, 8, 5, 17, 0, 17, 8, 23\}$  e quindi i quattro blocchi

$$\begin{pmatrix} 2 \\ 8 \end{pmatrix} \quad \begin{pmatrix} 5 \\ 17 \end{pmatrix} \quad \begin{pmatrix} 0 \\ 17 \end{pmatrix} \quad \begin{pmatrix} 8 \\ 23 \end{pmatrix}.$$

Moltiplicando ogni blocco per  $M$ , otteniamo come risultato i quattro blocchi

$$\begin{pmatrix} 18 \\ 0 \end{pmatrix} \quad \begin{pmatrix} 17 \\ 4 \end{pmatrix} \quad \begin{pmatrix} 7 \\ 25 \end{pmatrix} \quad \begin{pmatrix} 1 \\ 25 \end{pmatrix},$$

vale a dire, la stringa  $\{18, 0, 17, 4, 7, 25, 1, 25\}$  che, tradotta in lettere, diventa "SAREHZBZ". Come si vede, anche in questo caso a lettera uguale non corrisponde lettera uguale e la stessa lettera può corrispondere a lettere diverse.

Lo pseudocodice per questo cifrario è:

**Cifrario 3.3:** CIFRARIO DI HILL

Sia  $m \geq 2$  un intero. Sia  $\mathcal{P} = \mathcal{C} = (\mathbb{Z}_{26})^m$  e sia

$$\mathcal{K} = \{\text{matrici invertibili } m \times m \text{ a coefficienti in } \mathbb{Z}_{26}\}.$$

Per una chiave  $K$ , si definisce

$$\begin{aligned} c_K(x) &= Kx \\ d_K(y) &= K^{-1}y, \end{aligned}$$

dove tutte le operazioni sono da intendersi in  $\mathbb{Z}_{26}$ .

Il cifrario di Hill può essere rafforzato aumentando la dimensione della matrice ed usando matrici con parametri, ma non entriamo nei dettagli.

Sebbene questo cifrario possa essere resistente sia ad attacchi a forza bruta che ad attacchi di tipo statistico, purtroppo cade facilmente sotto un attacco di tipo *known plaintext*, vale a dire un attacco in cui il crittoanalista riesce a far spedire un messaggio da lui generato, così che egli conosce tanto il messaggio in chiaro che il messaggio cifrato. In questo modo, con un semplice sistema lineare è in grado di determinare la matrice  $M$ , che è la chiave.

Si può iniziare a pensare, a questo punto, che effettivamente ha ragione il protagonista de *Lo scarabeo d'oro*: per quanto si possa creare un cifrario complicato, applicandosi a fondo lo si può sempre violare senza ricorrere alla forza bruta, l'unico metodo sempre disponibile. In realtà, non è così: esiste un cifrario che è perfetto, nel senso che non è violabile se non tramite l'utilizzo della forza bruta, cosa che non è fattibile praticamente se lo spazio delle chiavi è sufficientemente ampio. Per illustrare questo cifrario, supporremo che il messaggio da cifrare sia una successione  $X$  di  $n$  bit. La cifratura avviene su ogni singolo bit: è quindi un cifrario a flusso. La chiave  $K$  è essa stessa una successione **assolutamente casuale** di  $n$  bit. Sia  $x_i$  l' $i$ -esimo bit del messaggio in chiaro e  $k_i$  l' $i$ -esimo bit della chiave. Allora, l' $i$ -esimo bit  $y_i$  del messaggio cifrato  $Y$  è definito semplicemente come

$$y_i = x_i \oplus k_i$$

dove  $\oplus$  è la somma binaria (cioè in  $\mathbb{Z}_2$ ) tra due bit (in termini informatici, è l'**OR esclusivo**, o **XOR**, tra due bit). La decodifica del messaggio si fa esattamente nello stesso modo:

$$x_i = y_i \oplus k_i$$

e questo perché

$$y_i \oplus k_i = (x_i \oplus k_i) \oplus k_i = x_i \oplus (k_i \oplus k_i) = x_i \oplus 0 = x_i,$$

dato che  $k_i \oplus k_i = 0$  indipendentemente dal valore di  $k_i$ . L'algoritmo, tra l'altro, risulta anche molto efficiente, come implementazione. Lo pseudocodice relativo è

**Cifrario 3.4:** CIFRARIO PERFETTO (ONE TIME PAD)

Sia  $x = (x_1, \dots, x_n)$  un messaggio,  $x_i \in \mathbb{Z}_2$ .

Sia  $\mathcal{P} = \mathcal{C} = \mathcal{K} = (\mathbb{Z}_2)^n$ .

Per  $K \in \mathcal{K}$ ,  $c_K(x)$  è definito come la somma dei due vettori  $x + K$ .

La decifratura si definisce nello stesso modo:  $d_K(y) = y + K$ . Si ha quindi

$$c_K(x) = (x_1 + K_1, \dots, x_n + K_n) \pmod{2}$$

$$d_K(y) = (y_1 + K_1, \dots, y_n + K_n) \pmod{2}$$

se  $K = (K_1 \dots K_n)$  e  $y = (y_1, \dots, y_n)$ .

La domanda sorge spontanea: se questo cifrario è perfetto, che bisogno c'è di studiare altri cifrari? Perché non si usa sempre questo? I motivi che indeboliscono questo cifrario sono i seguenti:

1. la chiave deve essere usata per cifrare un solo messaggio e poi gettata;
2. la chiave deve essere una successione realmente casuale di bit;
3. la chiave deve essere lunga tanto quanto il messaggio.

Esaminiamo il perché di ciò. Se la chiave fosse usata più di una volta, sarebbe molto semplice riuscire a determinare la chiave con dei semplici **XOR**, soprattutto nel caso in cui si riesca ad effettuare un attacco di tipo *known plaintext*. Accettato questo punto, ci troviamo di fronte al problema di generare una successione, possibilmente anche molto lunga, di bit casuali. E per casuali intendiamo realmente casuali, non generati al computer con una classica funzione di generazioni di numeri pseudocasuali. Tali funzioni, infatti, usando algoritmi matematici generano successioni di numeri che *sembrano* casuali ma, in realtà, non lo sono.

Si vede, quindi, che anche il solo generare una chiave diventa una problema significativo; ma c'è un altro problema, ancora più grosso: la gestione della chiave! Infatti, per decifrare è necessaria la stessa chiave usata per cifrare. Questo vuol dire che, a meno che la chiave non sia stata stabilita in precedenza, il destinatario oltre al messaggio dovrà ricevere anche la chiave. D'altra parte, il messaggio cifrato può essere trasmesso liberamente, ma la chiave no: per la chiave è necessario un canale che sia assolutamente sicuro. Ora, se disponiamo di un canale così sicuro da poter spedire la chiave, che

è lunga tanto quanto il messaggio, a che pro cifrare il messaggio? Tanto vale spedire il messaggio stesso tramite questo canale!

Si vede quindi che tutte queste problematiche limitano fortemente l'ambito entro il quale un cifrario di questo tipo può essere utilizzato.

### 3.2 Cifrari asimmetrici

Dalla discussione fin qui fatta, dovrebbe essere abbastanza chiaro che, al di là dell'algoritmo, buona parte della segretezza nella trasmissione di un messaggio cifrato dipende dalla chiave (secondo il Principio di Kerckhoffs) e dalla gestione della chiave. Infatti in un cifrario simmetrico per decifrare il messaggio è necessaria la stessa chiave usata per cifrarlo. Questo vuol dire che il destinatario deve essere in possesso della stessa chiave usata dal mittente. Se in alcune applicazioni (si pensi allo spionaggio, per esempio) può essere ammissibile che le due parti si incontrino precedentemente per concordare una o più chiavi da usare per le comunicazioni successive, ciò non è immaginabile nelle applicazioni quotidiane, soprattutto relativamente al commercio elettronico. Per esempio, se ci si vuole collegare telematicamente alla propria banca, non è immaginabile pensare di avere già pronte tante chiavi quanti sono gli utenti e le sessioni possibili. Nè si può generare al volo una chiave, perché sarebbe trasmessa su di un canale insicuro.

Nasce quindi l'esigenza di un tipo diverso di algoritmo, in cui siano necessarie due chiavi, una per cifrare ed una per decifrare e che siano ovviamente legate tra loro da qualche proprietà ma non deve essere possibile ricavare una delle due solamente conoscendo l'altra. In questo modo, se indichiamo le chiavi con  $k_1$  e  $k_2$ , con  $x$  il messaggio in chiaro, con  $y$  il messaggio cifrato, con  $c_{k_1}$  l'algoritmo per criptare tramite la chiave  $k_1$  e con  $d_{k_2}$  l'algoritmo per decriptare tramite  $k_2$ , è possibile immaginare un protocollo di comunicazione di questo tipo: la chiave  $k_1$  viene messa nel pubblico dominio, a disposizione di chiunque (e, per l'appunto, viene detta *chiave pubblica* dell'utente A); la chiave  $k_2$  viene custodita gelosamente (*chiave privata* di A). Per inviare un messaggio cifrato ad A, si cifra il messaggio  $x$  tramite  $k_1$ :  $y = c_{k_1}(x)$ . A questo punto, solo il legittimo destinatario, che è l'unico a conoscere  $k_2$ , è in grado di decifrare  $y$  tramite  $x = d_{k_2}(y)$ .

Oltre che per cifrare, tale algoritmo può essere usato anche per *autenticare* (firma digitale). Se A vuole certificare che è lui il generatore di un certo messaggio  $x$ , allora insieme ad  $x$  distribuisce la sua codifica  $y = c_{k_2}(x)$ : solo A può aver generato  $y$  perché è l'unico a conoscere  $k_2$ . Un'altra persona che vuole effettuare una verifica, non deve far altro che decifrare usando la chiave pubblica di A, cioè calcolare  $d_{k_1}(y)$  e verificare che il risultato coincide con  $x$ .

Esaminiamo uno dei più utilizzati algoritmi a chiave pubblica: l'algoritmo RSA. Per generare le due chiavi, pubblica e privata, è necessario deter-

minare due numeri primi  $p, q$ . Da questi, si costruisce il numero  $n = pq$  e poi si sceglie un numero  $a$  tale che  $\text{MCD}(a, (p-1)(q-1)) = 1$ . Dato che  $a$  è primo con  $\varphi(n) = (p-1)(q-1)$ , si può trovare grazie all'algoritmo euclideo  $b$  tale che  $ab \equiv 1 \pmod{n}$ . A questo punto, la chiave pubblica è definita dalla coppia  $k_1 = \{n, a\}$ , mentre la chiave privata è definita dalla terna  $k_2 = \{p, q, b\}$ . Descriviamo prima come funziona l'algoritmo, poi vediamo perché funziona ed infine perché è sicuro.

Il messaggio da cifrare  $x$  consiste in un numero primo con  $n$ . Per cifrare, si definisce

$$c_{k_1}(x) \equiv x^a \pmod{n}.$$

Per decifrare, si usa lo stesso procedimento usando la chiave privata:

$$d_{k_2}(y) \equiv y^b \pmod{n}.$$

Lo pseudocodice relativo è

**Cifrario 3.5:** CIFRARIO RSA

Sia  $n = pq$  dove  $p$  e  $q$  sono due primi. Sia  $\mathcal{P} = \mathcal{C} = \mathbb{Z}_n$  e

$$\mathcal{K} = \{(n, p, q, a, b) : ab \equiv 1 \pmod{\varphi(n)}\}.$$

Per una chiave  $K = (n, p, q, a, b)$ , si definisce

$$\begin{aligned} c_K(x) &= x^b \pmod{n} \\ d_K(y) &= y^a \pmod{n}. \end{aligned}$$

I valori  $(n, b)$  formano la chiave privata ed i valori  $(p, q, a)$  formano la chiave pubblica.

Si ottengono effettivamente due funzioni che sono una l'inversa dell'altra perché  $ab \equiv 1 \pmod{\varphi(n)}$  e quindi  $ab = 1 + k\varphi(n)$ , per qualche intero  $k$ , per cui, grazie al Teorema di Eulero 2.18,

$$\begin{aligned}
d_{k_2}(c_{k_1}(x)) &\equiv (c_{k_1}(x))^b \pmod{n} \\
&\equiv (x^a)^b \pmod{n} \\
&\equiv x^{ab} \pmod{n} \\
&\equiv x^{1+k\varphi(n)} \pmod{n} \\
&\equiv xx^{k\varphi(n)} \pmod{n} \\
&\equiv x(x^{\varphi(n)})^k \pmod{n} \\
&\equiv x1^k \pmod{n} \\
&\equiv x \pmod{n}.
\end{aligned}$$

L’algoritmo, quindi funziona. Ma perché è sicuro?

Per ottenere un algoritmo sicuro è necessario scegliere i numeri in gioco secondo alcuni criteri: i numeri primi  $p, q$  devono essere lunghi, in binario, almeno 512 bit (circa 154 cifre decimali) e anche il numero  $a$  deve essere scelto in modo che  $a > \sqrt[4]{n}$ .

L’algoritmo RSA si basa sul fatto che non esistono algoritmi noti “sufficientemente veloci” per determinare la fattorizzazione di un numero in numeri primi, né per risolvere il problema del logaritmo discreto, mentre esistono algoritmi “sufficientemente veloci” per fare tutte le altre operazioni usate: determinare due numeri primi “grandi” (test di Miller–Rabin), fare l’elevamento a potenza anche con numeri grandi modulo un altro numero (algoritmo “Square and Multiply”), calcolare il MCD (algoritmo euclideo), calcolare l’inverso di un numero modulo un altro numero (identità di Bézout, calcolabile tramite l’algoritmo euclideo esteso).

Questo vuol dire che tutte le operazioni descritte nel cifrario RSA sono eseguibili velocemente ed efficientemente. Per contro, se si volesse attaccare il cifrario partendo dalla chiave pubblica  $k_1 = \{n, a\}$ , bisognerebbe trovare la fattorizzazione  $n = pq$  in modo da poter poi calcolare  $b$  tramite l’algoritmo euclideo. Il problema però di fattorizzare  $n$  risulta essere un problema formidabile che, allo stato dell’arte degli algoritmi di fattorizzazione, non è possibile risolvere in un tempo virtualmente finito se i numeri hanno le dimensioni descritte.

Per avere informazioni aggiornate sullo stato dell’arte della fattorizzazione dei numeri e sui numeri primi, si può consultare il sito dei laboratori RSA: <http://www.rsasecurity.com/rsalabs/>

Nonostante l’algoritmo RSA sia considerato sicuro, non è, in realtà, l’algoritmo che viene di fatto usato nelle applicazioni per cifrare una comunicazione riservata e questo perché ha un tempo di esecuzione più lento rispetto ad altri cifrari simmetrici di pari sicurezza, come per esempio l’AES. Questi ultimi, quindi, si prestano meglio alla cifratura di un flusso di dati in cui è

necessario criptare una gran quantità di informazioni nell'unità di tempo. Il modo (un po' semplificato, in verità) in cui RSA viene usato è il seguente: se si vuole stabilire una comunicazione cifrata tra A e B, A individua la chiave pubblica di B e, tramite l'algoritmo RSA, invia a B una chiave segreta casuale generata per l'occasione e da usarsi una sola volta, per un cifrario simmetrico (per esempio, AES). A questo punto B tramite la sua chiave privata decifra il messaggio di A, recupera la chiave segreta e quindi i due interlocutori sono entrambi in possesso di una chiave per un cifrario simmetrico ed instaurano il flusso cifrato tramite tale algoritmo.

Gli algoritmi asimmetrici vengono quindi principalmente usati per una gestione sicura delle chiavi degli algoritmi simmetrici, oppure anche, come accennato, per definire dei protocolli di firma digitale.

## Riferimenti bibliografici

- [1] L. N. Childs, *A concrete introduction to higher algebra*, second edition, Springer UTM, 2000
- [2] D. R. Stinson, *Cryptography — Theory and practice*, second edition, Chapman & Hall / CRC, 2002

## Indice analitico

- Adleman, 17
- AES, 4, 18
- algoritmo
  - euclideo, 5–7
  - esteso, 7, 8
  - Miller–Rabin, 15, 17
  - square and multiply, 12, 13
- Bézout
  - identità di, 7, 12
- cifrari
  - asimmetrici, 26–29
    - chiave privata, 26
    - chiave pubblica, 26
  - simmetrici, 17–26
  - simmetrici ed asimmetrici, 17
- cifrario
  - a blocchi, 21
  - a flusso, 21
  - a sostituzione monoalfabetica, 19
  - a sostituzione polialfabetica, 20
  - di Cesare, 18
  - di Hill, 21–23
  - di Vigenère, 20
  - RSA, 27
- cifrato
  - One Time Pad, 25
- classi di congruenze, 11
  - operazioni con le, 12
- congruenza, 9, 10
- DES, 3, 18
- Diffie–Hellman, 17
- divisione euclidea, 5
- DSA, 3, 17
- ElGamal, 3, 17
- Enigma, 18
- Eulero
  - $\varphi$  di, 14
- firma digitale, 26
- IBM, 4
- Kerckhoffs
  - principio di, 18
- legge di cancellazione, 11
- massimo comune divisore (MCD), 4, 6, 8, 9
- minimo comune multiplo (mcm), 9
- NSA, 4
- potenze grandi, 12
- Rijndael, 18
- Rivest, 17
- RSA, 3, 17, 26, 28
- SHA-1, 17
- Shamir, 17
- Teorema
  - di Euclide, 5
  - di Eulero, 14
  - di Fermat, 15
  - di Miller–Rabin, 15
  - Miller–Rabin, 16
- test di primalità, 15
  - di Miller–Rabin, 15
  - probabilistico, 15