Crittografia

Andrea Centomo, Enrico Gregorio, Francesca Mantese

18 aprile 2007

Commenti, rilievi, segnalazioni di errori in vista di prossime edizioni possono essere segnalati agli autori

Prof. Andrea Centomo — Liceo Statale "F. Corradini" di Thiene (VI)
e-mail: `andrea.centomo@istruzione.it`

Prof. Enrico Gregorio — Università di Verona — Dipartimento di Informatica
e-mail: `enrico.gregorio@univr.it`

Dott. Francesca Mantese — Università di Verona — Dipartimento di Informatica
e-mail: `francesca.mantese@univr.it`

Indice

Prefazione

Progetto Lauree Scientifiche e Matematica

Da qualche decennio in tutte le nazioni industrializzate – l'Italia non fa eccezione – il reclutamento dei giovani scienziati è in crisi. Nel 2003 il nostro Ministero dell'Istruzione e dell'Università ha convocato i Presidi delle Facoltà di Scienze e i rappresentanti di Confindustria per mettere a punto il Progetto Lauree Scientifiche, un'iniziativa da sviluppare, a cura delle Università, nelle Scuole Secondarie italiane, per la promozione delle discipline scientifiche. Il Progetto, almeno inizialmente, coinvolge solo una modesta porzione della popolazione scolastica, puntando alla qualità più che ai grandi numeri. Affinché i risultati si possano diffondere, saranno necessarie altre iniziative e torneranno molto utili i materiali didattici del tipo della presente pubblicazione. Rispettando criteri nazionali comuni, il PLS è realizzato per discipline separate e su base regionale, in collaborazione tra Università e Ufficio Scolastico. Nel Veneto, per la Matematica, sono stati scelti 15 istituti secondari, un paio per provincia. In ciascun "polo" un gruppo di progetto, composto da tre docenti della scuola secondaria e due universitari, ha organizzato una sorta di "Laboratorio matematico" con queste modalità: un piccolo gruppo di studenti volontari (15–20) viene riunito in orario extrascolastico, circa cinque riunioni di un paio d'ore ciascuna. Ai giovani viene proposto un problema concreto, nella cui soluzione la matematica svolge un ruolo determinante. Dopo una breve introduzione dei docenti, i ragazzi si mettono al lavoro, penna e computer, per elaborare una soluzione.

Il problema matematico da trattare è stato scelto in ciascun polo in modo autonomo. Il tema Crittografia, proposto dal collega prof. Alessandro Languasco dell'Università di Padova, ha avuto la preferenza di cinque poli su quindici. Argomento di tradizione antica, la Crittografia è di grande attualità, indispensabile nel delicato problema del trattamento dei dati riservati (per esempio, il bancomat). Dal punto di vista matematico, l'argomento ha il pregio di non richiedere pesanti prerequisiti e, d'altra parte, di introdurre rapidamente a problemi di ricerca tuttora aperti. Nel polo di Schio-Thiene (Licei "N. Tron" e "F. Corradini") l'attività dei Labo-

ratori ha dato l'occasione di predisporre eccellente materiale didattico, che è stato raccolto nel presente fascicolo a cura dei professori Andrea Centomo (del Liceo "F. Corradini"), Enrico Gregorio e Francesca Mantese (dell'Università di Verona). La pubblicazione di questo materiale metterà in grado altri docenti della Scuola Secondaria di ripetere l'esperimento nelle loro classi, nel preciso spirito del Progetto Lauree Scientifiche. Siamo molto grati agli Autori, che per questa realizzazione hanno messo a disposizione competenza, tempo ed entusiasmo.

Prof. Benedetto Scimemi
Coordinatore scientifico P.L.S. (Matematica) per il Veneto.

Confindustria e il P.L.S.

Confindustria Vicenza ha aderito al progetto Lauree Scientifiche attraverso la sottoscrizione fatta dalla propria rappresentanza regionale al Protocollo d'intesa definito con alcune Università venete nel febbraio 2006.

Condiviamo infatti pienamente l'obiettivo del Progetto, quello cioè di promuovere e sostenere iniziative che mirano all'incremento degli immatricolati e dei laureati nelle materie scientifiche – in relazione anche alle raccomandazioni dell'Unione Europea – e di un maggior raccordo con la domanda di lavoro.

Scienza e tecnologia, mai come nel secolo appena trascorso, hanno così profondamente inciso sulla nostra vita, eppure negli anni il numero totale degli iscritti ai corsi di laurea di Matematica, Fisica e Chimica ha registrato una diminuzione, anche se recentemente si assiste ad una piccola ripresa. Invece un alto grado di pervasività della cultura scientifica nell'ambiente scolastico, nella vita e nelle imprese favorisce lo sviluppo e rende il Paese più dinamico e competitivo: i settori del futuro sono infatti tutti caratterizzati da alte conoscenze tecnico-scientifiche ed investire nella scienza è una scelta saggia per gli studenti e per le imprese.

La scienza quindi deve vivere nelle nostre scuole, nelle nostre imprese e nella nostra società. È necessario allora non solo spingere le imprese a puntare di più sui giovani talenti e a valorizzare i laureati nelle discipline scientifiche, ma anche far crescere le vocazioni scientifiche tra i giovani, trasmettendo già durante la Scuola Superiore l'interesse e la passione verso materie certamente impegnative ma anche ricche di contenuti formativi ed illustrare loro gli sbocchi occupazionali che una buona preparazione scientifica e tecnologica apre.

Confindustria Vicenza quindi è impegnata a sostenere tutte quelle iniziative – e tale è il presente volumetto – che si pongono l'obiettivo di far capire ai giovani come una laurea a carattere scientifico possa dare un tipo di preparazione e di atteggiamento mentale che permette un inserimento lavorativo di successo in vari contesti lavorativi.

Giuseppe Filippi
Coordinatore della Commissione Scuola – Confindustria Vicenza

Introduzione

In queste pagine sono stati raccolti, riorganizzati e arricchiti i materiali didattici utilizzati nel corso del Progetto Lauree Scientifiche (PLS) per il Veneto dal nostro gruppo[1] di lavoro.

L'argomento trattato è la crittografia a chiave pubblica. Il motivo di questa scelta deriva dal fatto che le conoscenze matematiche necessarie per comprendere il funzionamento di rilevanti codici di crittografia, a livello non superficiale, sono elementari. Bastano poche ore per spiegare a uno studente delle scuole superiori ben intenzionato il funzionamento del sistema crittografico RSA e i prerequisiti che egli deve possedere sono solo i concetti di numero primo, scomposizione in fattori primi e massimo comun divisore.

Recentemente sono state pubblicate diverse monografie che hanno per oggetto la crittografia. Tuttavia, lo scopo di questa breve raccolta è di fornire un'esposizione elementare dell'argomento, ma che non rinunci a un impianto rigoroso e formale. Ciò al fine di permettere allo studente della scuola superiore, cui si rivolgono in modo particolare queste pagine, di formarsi un'idea del modo in cui si opera in matematica. I contenuti di Crittografia per Studenti possono risultare utili anche all'insegnante desideroso di disporre di materiali didattici sperimentati sul campo da cui procedere per eventuali adattamenti nella prassi didattica curriculare o per la progettazione di un piccolo corso monografico sul tema della crittografia.

Questa raccolta è suddivisa in quattro capitoli cui segue un capitolo che contiene alcuni complementi. Nel primo capitolo si discutono le principali proprietà aritmetiche dell'insieme dei numeri interi. Nel secondo sono contenute le nozioni di aritmetica modulare essenziali per poter affrontare lo studio della crittografia a chiave pubblica. Nel terzo capitolo vengono discusse la crittografia classica e la crittografia a chiave pubblica, con particolare attenzione al crittosistema RSA. Nel

[1] Il gruppo si è costituito a Padova nel settembre del 2005 con una composizione mista di docenti dell'Università di Verona (Dott.ssa Francesca Mantese e Prof. Enrico Gregorio), del Liceo "N. Tron" di Schio (Prof. Giorgio Pizzolato, Prof.ssa Sabrina Zambon), del Liceo "G. Galilei" di Verona (Prof. Sandro Pistori) e del Liceo "F. Corradini" di Thiene (Prof. Andrea Centomo). Nell'anno scolastico 2006-2007 è stata aggregata al progetto la Prof.ssa Anna Bettale del Liceo "N. Tron".

quarto capitolo vengono descritti i principali comandi dell'ambiente per la crittografia GPG sviluppato come alternativa *open source* al pionieristico ambiente PGP elaborato da P. Zimmerman. Il software GPG permette a comuni utenti di personal computer di poter usufruire sul loro sistema di uno strumento di crittografia forte. Infine, in diversi capitoli viene utilizzato il software *open source* per la Teoria dei Numeri PARI GP, che permette di trattare esempi significativi di crittografia e di aritmetica sia attraverso un buon numero di funzioni predefinite che attraverso un ambiente di scripting. Il capitolo quinto contiene una rielaborazione delle tesine presentate da alcuni studenti nel secondo anno del Progetto PLS.

Crediti

I materiali didattici a cui devono la nascita queste pagine sono stati tutti forniti dal prof. Alessandro Languasco dell'Università di Padova, nella forma di dispense ora pubblicate nell'ambito del Progetto Lauree Scientifiche [8]. A ciò si devono aggiungere due articoli molto interessanti: il primo di Giovanni Alberti [1] in cui viene documentato un percorso didattico di crittografia per studenti delle scuole medie superiori, il secondo di Alessandro Zaccagnini [10] sui numeri primi e l'algoritmo di Elgamal.

Nel capitolo quinto sono state inserite le sezioni maggiormente significative dei seguenti lavori:

- Poligoni stellati e Teorema di Wilson, redatto da Chiara Coriele del Liceo "N. Tron" di Schio;
- Implementazione in Javascript del metodo di Vigènere, redatto da Alessandro Dal Maso del Liceo Classico "F. Corradini" di Thiene;
- Il Teorema dei Numeri Primi, redatto da Letizia Saugo del Liceo Linguistico "F. Corradini" di Thiene.

Ringraziamenti

Si ringrazia la dottoressa Lucia Gecchelin per aver letto e corretto il manoscritto.

Capitolo 1

Aritmetica in $\mathbb{Z}$

Lo studio della crittografia a chiave pubblica, che costituisce l'oggetto principale di questa raccolta, non può prescindere dalla conoscenza di un certo numero di risultati fondamentali di aritmetica, per la cui trattazione ci riferiamo principalmente a [5], mentre per le note storiche il riferimento è al classico [2].

Indichiamo con $\mathbb{Z}$ l'insieme dei numeri interi

$$\mathbb{Z} = \{0, 1, -1, 2, -2, \ldots\}$$

e con $\mathbb{Z}^*$ l'insieme dei numeri interi escluso 0. Nell'insieme dei numeri interi sono definite le operazioni di *addizione* e *moltiplicazione* che godono delle seguenti proprietà:

(a) proprietà associativa: $(a+b)+c = a+(b+c)$, $(ab)c = a(bc)$,

(b) proprietà commutativa: $a+b = b+a$, $ab = ba$,

(c) elemento neutro: $a+0 = a$, $1a = a$,

(d) opposto: per ogni a esiste $-a$ tale che $a+(-a) = 0$,

(e) proprietà distributiva: $a(b+c) = ab+ac$,

con $a, b, c \in \mathbb{Z}$. Le operazioni di addizione e moltiplicazione con le relative proprietà rendono $\mathbb{Z}$ ciò che algebricamente si dice un *anello commutativo*. Le proprietà che rendono $\mathbb{Z}$ un anello permettono di giustificare numerosi fatti algebrici di uso comune come ad esempio la legge di annullamento del prodotto, le regole dei segni e via di seguito.

Si ricorda poi che, dato un qualsiasi numero intero a, si definisce *valore assoluto*

di a il numero intero non negativo così definito

$$|a| = \begin{cases} a, & a \geq 0 \\ -a, & a < 0 \end{cases}$$

dove $-a$ indica l'*opposto* di a. Così, ad esempio, avremo che $|-3| = 3$ e che $|5| = 5$.

1.1 Numeri primi

Cominciamo richiamando il concetto di divisibilità.

Definizione 1. Dati due numeri interi a e b, con $b \neq 0$, si dice che b *divide* a se esiste un numero intero q tale che $a = bq$.

La notazione $b \mid a$ si legge "b divide a". Inoltre se $b \mid a$ si dice anche che b è un divisore o un fattore di a, oppure che a è un multiplo di b o ancora che a è divisibile per b. Osserviamo che dalla definizione precedente segue che nessun numero è divisibile per 0.

Esempio 2. L'intero -4 divide -28: infatti $-28 = -4 \cdot 7$. Analogamente $-7 \mid 0$ in quanto sappiamo che $0 = -7 \cdot 0$.

Risultano di verifica immediata i seguenti fatti:

(a) sia $a \in \mathbb{Z}^*$ allora $a \mid a$ e $-a \mid a$;

(b) sia $b \in \mathbb{Z}$ allora $1 \mid b$ e $-1 \mid b$.

Per questa ragione, dato $a \in \mathbb{Z}^*$, chiameremo *divisori banali* di a i numeri $\pm a$ e ± 1.

Definizione 3. Un intero a, con $|a| > 1$, si dice *primo* se ammette solo divisori banali.

Nota 4. Si osserva che la Definizione 3 esclude dall'insieme dei possibili numeri primi 0, 1 e -1. Per quanto visto al punto (a) precedente, l'esclusione dello 0 non appare sorprendente. Tuttavia potrebbero risultare oscuri i motivi che portano all'esclusione di ± 1 dall'insieme dei numeri primi. Il lettore interessato alle motivazioni di questa scelta può fare riferimento a [6].

Un primo problema che si può porre riguarda l'opportunità di introdurre il concetto di numero primo in matematica. La questione sembrerebbe oziosa non da ultimo in quanto oggi tutti sappiamo quanto siano importanti i numeri primi. Tuttavia, se i primi fossero tre o quattro in tutto i matematici probabilmente non si sarebbero interessati tanto alle loro proprietà. Invece, già Euclide[1] era a conoscenza della seguente notevole proprietà.

[1] L'elegante dimostrazione dell'infinità dei numeri primi si trova nel Libro IX (Proposizione 20) degli *Elementi* di Euclide.

Proposizione 5. *I numeri primi sono infiniti.*

Dimostrazione. Supponiamo che i numeri primi siano un numero finito e che siano tutti e soli

$$p_1, p_2, \ldots, p_n.$$

Considerato il prodotto $p = p_1 \cdot p_2 \cdots p_n$ il numero $p+1$ non è primo e quindi esiste un primo p_i tale che $p_i \mid (p+1)$. Ora osservato che $p_i \mid p$ si ha l'assurdo $p_i \mid 1$. □

Il concetto di numero primo ci permette di formulare il Teorema Fondamentale dell'Aritmetica[2].

Teorema 6 (Teorema Fondamentale dell'Aritmetica). *Ogni numero intero a, con $|a| > 1$, si può scrivere come prodotto di numeri primi. La fattorizzazione è unica a meno di ordine e segno dei fattori.*

Dimostrazione. Il lettore è rinviato a [5]. □

La determinazione della fattorizzazione di un numero in primi, nota anche con il nome di scomposizione in fattori primi, è un esercizio in cui tutti ci siamo cimentati, almeno nel caso di numeri interi positivi e con poche cifre. Ciò non è un caso in quanto, come vedremo in seguito, la fattorizzazione in primi rientra a oggi nel novero dei problemi computazionalmente a elevata complessità.

Esempio 7. Scomporre in fattori primi il numero -117. Osserviamo che -117 è divisibile per -3 in quanto $-117 : (-3) = 39$ e quindi notiamo che $39 = 3 \cdot 13$. In conclusione allora la scomposizione in fattori primi è $-117 = -3^2 \cdot 13$. Anche $-117 = -13 \cdot 3^2$ è una scomposizione in fattori primi di -117, tuttavia essa coincide con la prima a meno di ordine e segno dei fattori.

Ricorrendo al Teorema Fondamentale dell'Aritmetica possiamo introdurre i concetti di *minimo comune multiplo* e *massimo comun divisore*. Infatti dati due numeri interi a e b, non nulli, essi si possono scrivere nella forma

$$a = \pm p_1^{k_1} \cdot p_2^{k_2} \cdot \ldots \cdot p_n^{k_n} \qquad b = \pm p_1^{h_1} \cdot p_2^{h_2} \cdot \ldots \cdot p_n^{h_n}$$

dove i numeri p_1, p_2,..., p_n sono tutti primi *distinti* e positivi; se un primo non compare nella fattorizzazione di uno dei due numeri, l'esponente è zero. Scriviamo allora

$$(a,b) = p_1^{l_1} \cdot p_2^{l_2} \cdot \ldots \cdot p_n^{l_n} \qquad [a,b] = p_1^{g_1} \cdot p_2^{g_2} \cdot \ldots \cdot p_n^{g_n}$$

dove $l_1 = \min\{k_1, h_1\}$, $l_2 = \min\{k_2, h_2\}$, ..., $l_n = \min\{k_n, h_n\}$ e $g_1 = \max\{k_1, h_1\}$, $g_2 = \max\{k_2, h_2\}$, ..., $g_n = \max\{k_n, h_n\}$. Il numero intero positivo (a,b) prende il nome di

[2] La prima dimostrazione di questo teorema si trova nel Libro IX (Proposizione 14) degli *Elementi* di Euclide. Una dimostrazione rigorosa di esso è esposta nelle *Disquisitiones Arithmeticae* di K. F. Gauss del 1801.

massimo comun divisore tra a e b, mentre il numero intero positivo $[a,b]$ prende il nome di *minimo comune multiplo*. Si noti che $(a,b)[a,b] = ab$.

Esempio 8. Determinare minimo comune multiplo e massimo comun divisore tra -117 e 65. Osserviamo che si ha

$$-117 = -3^2 \cdot 5^0 \cdot 13 \qquad 65 = 3^0 \cdot 5 \cdot 13$$

da cui $(-117,65) = 3^0 \cdot 5^0 \cdot 13 = 13$ e $[-117,65] = 3^2 \cdot 5 \cdot 13 = 585$.

Esiste un modo alternativo per definire il massimo comun divisore tra due numeri interi che *non* richiede il concetto di scomposizione in fattori primi e che conduce all'identità di Bézout[3].

Teorema 9 (Divisione Euclidea). *Siano a e b due numeri interi con $b \neq 0$, allora esistono unici due numeri interi q e r tali che $a = qb + r$, con $0 \leq r < |b|$.*

Dimostrazione. Consideriamo l'insieme M formato dai multipli di $|b|$ che sono minori o uguali ad a ossia

$$M = \{x \in \mathbb{Z} : x = k|b|, x \leq a\}$$

e osserviamo che non è un insieme vuoto. Infatti $-|a| \in M$ in quanto $-|a||b| \leq -|a| \leq a$. Ora indichiamo con $h|b|$ il massimo di M e osserviamo che

$$h|b| \leq a \iff a = h|b| + r$$

con $r \geq 0$. D'altra parte si ha anche

$$(h+1)|b| = h|b| + |b| > h|b| \implies (h+1)|b| > a \implies (h+1)|b| > h|b| + r$$

e quindi $r < |b|$. Posto $q = h$ se $b > 0$ o $q = -h$ se $b < 0$, si ha $h|b| = qb$ e $a = qb + r$. □

I numeri q e r sono rispettivamente il *quoziente* e il *resto* della divisione euclidea[4] tra a e b.

Esempio 10. Determinare q ed r nel caso in cui $a = 637$ e $b = -47$. Innanzitutto osserviamo che

$$637 : 47 = 13$$

con il resto di 26. Quindi

$$637 = 47 \cdot 13 + 26 = -47 \cdot (-13) + 26$$

da cui $q = -13$ e $r = 26$.

[3] Così chiamata in onore del matematico francese E. Bézout (1730–1783) fu in realtà stabilita in precedenza dal matematico francese C. G. Bachet de Méziriac (1581–1638).

[4] Il procedimento di divisione euclidea è descritto *more geometrico* da Euclide nel Libro VII (Proposizioni 1 e 2) degli *Elementi*. Il contesto in cui Euclide illustra il procedimento riguarda il calcolo del massimo comun divisore tra due interi positivi.

Esempio 11. Determinare q ed r nel caso in cui $a = -637$ e $b = 47$. Per quanto visto all'esempio precedente, non è difficile vedere che

$$637 = 47 \cdot 14 - 21 \iff -637 = -14 \cdot 47 + 21$$

da cui $q = -14$ e $r = 21$.

Teorema 12 (Identità di Bézout). *Siano $a, b \in \mathbb{Z}^*$. Allora esistono due interi u e v tali che*

$$(a, b) = au + bv.$$

Dimostrazione. Consideriamo l'insieme di numeri interi

$$I = \{ax + by : x, y \in \mathbb{Z}\}$$

e osserviamo che non è vuoto in quanto, ad esempio, $|a| > 0$ vi appartiene. Sia ora $d \in I$ il minore tra tutti i numeri interi *positivi* di I e dimostriamo che $d = (a, b)$. Per il Teorema 9 si ha

$$b = dq + r$$

con $0 \leq r < d$. Inoltre si ha

$$r = b - dq = b - (au + bv)q = b(1 - vq) - auq$$

da cui vediamo che $r \in I$. Essendo d il minimo degli interi positivi di I ed essendo il resto non negativo deve essere $r = 0$. Allora $d \mid b$ e, in modo del tutto analogo, anche $d \mid a$. Supponiamo ora che esista un intero e tale che $e \mid a$ e $e \mid b$; allora $e \mid au$ e $e \mid bv$ e quindi $e \mid d$ in quanto $d = au + bv$. Quindi d è proprio il massimo comun divisore. □

1.2 Algoritmo di Euclide

Esiste un terzo modo di natura algoritmica che permette il calcolo del massimo comun divisore tra due numeri interi e, nella sua versione estesa, il calcolo dei coefficienti che compaiono nell'identità di Bézout.

Lemma 13. *Se $a = qb + r$ allora $(a, b) = (b, r)$.*

Dimostrazione. Sia $d = (a, b)$ allora $a = kd$ e $b = hd$ e quindi, osservato che

$$r = a - qb = kd - qhd = (k - qh)d$$

si ha che $d \mid r$. Inoltre $d = (b, r)$ in quanto se esistesse $d' > d$, con $d' \mid b$ e $d' \mid r$, avremo che $d' \mid a$ e quindi arriveremo all'assurdo per cui $d' = (a, b)$. □

Il Lemma 13 suggerisce il seguente procedimento, che va sotto il nome di *algoritmo di Euclide*[5], per il calcolo del massimo comun divisore tra due numeri.

Siano $a, b \in \mathbb{Z}$, con $a > b$, il calcolo del loro massimo comun divisore (a, b) con l'algoritmo di Euclide consiste nei seguenti passaggi:

- dividiamo a per b ottenendo $a = bq_1 + r_1$, con $0 \leq r_1 < b$. Se $r_1 = 0$ allora $(a, b) = b$, altrimenti
- dividiamo b per r_1 ottenendo $b = r_1 q_2 + r_2$, con $0 \leq r_2 < r_1$. Se $r_2 = 0$ allora $(a, b) = r_1$, altrimenti
- dividiamo r_1 per r_2 ottenendo $r_1 = r_2 q_3 + r_3$, con $0 \leq r_3 < r_2$. Se $r_3 = 0$ allora $(a, b) = r_2$, altrimenti si continua.

Osservato che al procedere del numero di iterazioni il resto della divisione decresce, deve esistere un numero k tale che $r_k = 0$, giunti a questo punto avremo $(a, b) = r_{k-1}$.

Esempio 14. Calcolare con l'algoritmo di Euclide $(1071, 1029)$. Dalla seguente tabella si ha:

1071	1029	$1071 = 1 \cdot 1029 + 42$	$q_1 = 1$	$r_1 = 42$
1029	42	$1029 = 24 \cdot 42 + 21$	$q_2 = 24$	$r_2 = 21$
42	21	$42 = 2 \cdot 21 + 0$	$q_3 = 2$	$r_3 = 0$

e quindi, osservato che l'ultimo resto non nullo è 21, $(1071, 1029) = 21$.

1.2.1 Algoritmo di Euclide esteso

Iniziamo con un esempio, supponendo che l'algoritmo di Euclide termini dopo tre iterazioni:

a	b	$a = bq_1 + r_1$	q_1	$r_1 = a - bq_1$
b	r_1	$b = r_1 q_2 + r_2$	q_2	$r_2 = b - r_1 q_2$
r_1	r_2	$r_1 = r_2 q_3$	q_3	$r_3 = 0$

Allora, sostituendo a ritroso, avremo:

$$r_1 = r_2 q_3 = q_3(b - r_1 q_2) = q_3 b - q_3 q_2 (a - bq_1) = \\ = q_3[(1 + q_2 q_1)b - q_2 a] = q_3(ua + vb)$$

dove si è posto $u = -q_2$ e $v = (1 + q_2 q_1)$. Se ora ricordiamo che r_2 è uguale a (a, b) avremo anche $(a, b) = ua + vb$ che rappresenta proprio l'identità di Bézout vista

[5] Il procedimento è descritto da Euclide nel Libro VII (Proposizioni 1 e 2) degli *Elementi*.

in precedenza. Non è difficile comprendere che un ragionamento del tutto analogo si può applicare nel caso in cui l'algoritmo di Euclide termini dopo un numero maggiore di passi. Un esempio concreto forse può chiarire ulteriormente il procedimento.

Esempio 15. Determinare i coefficienti di Bézout di $(1071, 1029)$. Sfruttando a ritroso i calcoli dell'Esempio 14 si ha:

$$\begin{aligned} 21 &= 1029 - 24 \cdot 42 = 1029 - 24 \cdot (1071 - 1029) = \\ &= -24 \cdot 1071 + 25 \cdot 1029 \end{aligned}$$

da cui $(1071, 1029) = -24 \cdot 1071 + 25 \cdot 1029$ e quindi $u = -24$ e $v = 25$.

1.3 Crivello di Eratostene

Nel paragrafo precedente abbiamo visto che i numeri primi sono infiniti e come tramite essi si possa determinare la scomposizione in fattori di un numero intero. Vi è tuttavia un problema, che abbiamo volutamente trascurato, ma su cui è necessario discutere: come si determinano i numeri primi?

Nel corso dei secoli sono stati fatti numerosi tentativi per definire un algoritmo capace di generare solo numeri primi e sono state avanzate numerose congetture sulla primalità di numeri appartenenti a determinate classi. Alcune di queste congetture sono state in seguito smentite, altre sono ancora da dimostrare.

Il sistema più semplice per ricercare i numeri primi minori di un certo numero fissato è stato ideato dal matematico greco Eratostene[6] e prende il nome di *crivello o setaccio di Eratostene*. Vediamo praticamente come esso funzioni nel caso specifico della determinazione dei numeri primi da 2 a 50.

	2	**3**	4	**5**	6	**7**	8	9	10
11	12	**13**	14	15	16	**17**	18	**19**	20
21	22	**23**	24	25	26	27	28	**29**	30
31	32	33	34	35	36	**37**	38	39	40
41	42	**43**	44	45	46	**47**	48	49	50

Tabella 1.1: Crivello di Eratostene

Si scrivono i numeri da 2 fino a 50 in ordine crescente. Quindi si individua il primo numero primo **2** e si eliminano tutti i numeri pari, che non sono primi in quanto multipli di 2. Si individua il secondo numero primo **3** e si eliminano tutti i

[6] Eratostene di Cirene (276 a.C.–194 a.C.) matematico, astronomo, geografo e poeta alessandrino, noto soprattutto per aver misurato le dimensioni della Terra con grande precisione.

suoi multipli. Si procede similmente per tutti i numeri interi inferiori a $\sqrt{50}$ (quindi fino a 7). Il risultato si vede nella Tabella 1.1. La tecnica del crivello di Eratostene può sembrare rudimentale tuttavia a volte rappresenta, tuttora, l'unico metodo per trovare un numero primo. Naturalmente i setacci impiegati attualmente sono più sofisticati di questo, in quanto in essi si introducono dei trucchi per ridurre al minimo il numero di calcoli da eseguire.

1.4 Un primo approccio a PARI GP

Come suggerisce il nome, PARI GP è una combinazione di due ambienti:

1. PARI: una libreria di routine scritte in C, orientata ad applicazioni di Teoria dei Numeri, caratterizzata da una notevole velocità di esecuzione;
2. GP: un interprete che consente l'accesso alla libreria estesa PARI attraverso una serie di comandi da terminale. Ciò consente di disporre di un semplice ambiente di programmazione nel linguaggio di scripting[7] GP.

PARI GP è software *libero* (rilasciato con licenza GPL) e quindi usabile e ridistribuibile per qualsiasi scopo. Il fondatore del progetto PARI GP è H. Cohen dell'Università di Bordeaux 1 e il software è attualmente mantenuto da K. Belabas della stessa Università. Trattandosi di software *libero* è importante ricordare che tutte le informazioni e la documentazione per l'uso del programma sono disponibili in rete e, in particolare, al sito ufficiale del progetto

`http://pari.math.u-bordeaux.fr/`

Per lanciare PARI è sufficiente aprire un terminale e digitare `gp`. Dopo un attimo verrà visualizzata la schermata di benvenuto che si può vedere nella Tabella 1.2.

Osserviamo in primo luogo che tramite il comando `?` si può accedere al *manuale in linea* e alle sue diverse sezioni, mentre per uscire dal programma è sufficiente digitare il comando `\q`. Allo startup viene allocata una certa quantità di memoria e vengono eseguiti dei calcoli preliminari. Le variabili `parisize` e `primelimit` indicano rispettivamente che sono stati allocati, per il funzionamento del programma, 4000000 byte e che sono stati calcolati tutti i numeri primi fino a 500000. Questi valori influenzano il comportamento di diverse funzioni aritmetiche disponibili. Se tali valori non fossero sufficienti è possibile

(a) eseguire GP ricorrendo al comando `gp -s NUMERO` oppure, durante l'esecuzione di GP, usare il comando `allocatemem` che raddoppia la quantità di memoria preesistente;

[7] Per chi lo desidera è disponibile il compilatore GP2C che consente di trasformare uno script GP in un codice C.

```
            GP/PARI CALCULATOR Version 2.3.0 (released)
      i686 running linux (ix86/GMP-4.1.4 kernel) 32-bit version
             compiled: Dec 29 2006, gcc-4.1.1 20061011
         (readline v5.1 enabled, extended help available)

              Copyright (C) 2000-2006 The PARI Group

PARI/GP is free software, covered by the GNU General Public
License, and comes WITHOUT ANY WARRANTY WHATSOEVER.

Type ? for help, \q to quit.
Type ?12 for how to get moral (and possibly technical) support.

parisize = 4000000, primelimit = 500000
?
```

Tabella 1.2: Schermata di benvenuto di PARI GP.

(b) eseguire GP ricorrendo al comando `gp -p NUMERO` per aumentare il numero di primi precalcolati.

1.4.1 Esempi elementari

Iniziamo risolvendo alcuni semplici esercizi di aritmetica mediante delle funzioni predefinite in PARI.

Esempio 16. *Posto* $x = 3456789$ *calcolare il massimo comun divisore tra* 23457 *e* x^3.

Ricorrendo alla funzione predefinita `gcd` possiamo eseguire immediatamente questo calcolo:

```
? x=3456789
%1 = 3456789

? gcd(23457,x^3)
%2 = 21
```

Il comando `x=3456789` assegna alla variabile `x` il corrispondente valore. Mentre la funzione predefinita `gcd` (greatest common divisor) permette il calcolo del massimo comun divisore cercato. Osserviamo che ogni risposta che PARI restituisce viene numerata attraverso un'etichetta del tipo `%n` dove `n` indica il numero di risposta.

Esempio 17. *Dire se il numero* 2354567 *è primo e in caso negativo scomporlo in fattori primi.*

Utilizzando la funzione predefinita `isprime` possiamo vedere se il numero dato è primo[8]:

```
? isprime(2354567)
%3 = 0
```

La risposta `0` è negativa. Possiamo allora procedere alla determinazione della scomposizione del numero in fattori primi utilizzando la funzione `factor`:

```
? factor(2354567)
%4 = [743 1]
     [3169 1]
```

da cui si ha che $235467 = 743 \cdot 3169$.

Esempio 18. *Con riferimento all'esempio 16 calcolare* x^{35}.

Possiamo calcolare immediatamente questa potenza:

```
%5 = 713757190005104555760075970012707062016650950 83
985448409911207166641615899749138577674825194469323 4
416610551776667867417303492792738770928430538536753 1
374382538407481036582959105398548157329655244526947 7
228450400641174421655031 49
```

L'esempio ci mostra come il programma acceda a una aritmetica intera a lunghezza arbitraria.

Esempio 19. *Scrivere uno script per il calcolo del massimo comun divisore di due numeri.*

Uno degli aspetti più interessanti dell'ambiente PARI GP è che esso può essere programmato includendo delle funzioni nuove definite dall'utilizzatore. Anche se la funzione predefinita `gcd` permette il calcolo del massimo comun divisore è istruttivo scrivere autonomamente uno script che esegua lo stesso calcolo. L'algoritmo che utilizziamo è l'algoritmo di Euclide descritto in precedenza.

Per implementare questo algoritmo procediamo per passi:

1. con un editor di testo per la programmazione (ad esempio Scite) scriviamo un codice GP che implementi l'algoritmo di Euclide; un esempio sarà dato di seguito;
2. salviamo il codice con nome `euclide.gp`;

[8] Per essere più precisi dovremmo osservare che `isprime(x,{flag=0})` se `flag=0` o se viene omesso esegue un controllo della primalità usando una combinazione di algoritmi. Specificando `flag=1` viene utilizzato l'algoritmo di Pocklington-Lehmer, mentre se `flag=2` si utilizza APRLC.

3. lanciamo PARI GP e digitiamo il comando `\r euclide.gp` in modo che il file venga letto;
4. scriviamo ad esempio `Euclide(1255,55)` per calcolare il massimo comun divisore tra i numeri 1255 e 55;
5. il risultato sarà 5 con 4 iterazioni.

Ecco un esempio di script per il calcolo:

```
/*****************************************************
*                 ALGORITMO EUCLIDEO
*       input: n,m - numeri di cui si calcola MCD
*       output: d - Massimo Comun Divisore tra m e n
*          A. CENTOMO per il PLS 2005-2007
*****************************************************/
{Euclide(m,n) = local (d ,app , a , b , iterazioni);
/* se n=m il risultato e' banale */
if(n == m, d = n; print("I due numeri sono uguali");
 print("d=",d);
 return;
 );
/* se m < n scambio i due numeri */
if(m < n, app=m; m=n; n=app);
while (n!=0,
     r= m %n ; /* resto della divisione intera di m per n */
     m=n; /* scambio il ruolo di m e n e di n e r */
     n=r; iterazioni=iterazioni+1 /*conto il numero di iterazioni*/
    );
 d=m;
print("Massimo Comun Divisore");
 print(d);
print("Iterazioni");
 print(iterazioni);}
```

Tramite la funzione predefinita `bezout` possiamo calcolare in PARI i coefficienti che compaiono nell'identità di Bézout e il massimo comun divisore tra due numeri assegnati:

```
? bezout(600,125)
%1 = [-1, 5, 25]
```

Infatti: $(600,125) = 25$ e $25 = 5 \cdot 125 - 600$.

Capitolo 2

Aritmetica Modulare

Introduciamo ora una nuova operazione tra numeri interi, che chiamiamo operazione *modulo* e che indichiamo con il simbolo %, definita da

$$a \,\%\, b = r$$

dove r indica il resto della divisione euclidea tra a e b. Chiaramente essendo $0 \leq r < b$ il risultato di questa operazione è sempre positivo e strettamente minore di b.

Nota 20. Questa operazione apparentemente astratta in realtà si usa più di quanto si possa credere. Ad esempio, quando alle ore 15 guardiamo l'orologio e vediamo che la lancetta delle ore segna le 3 non ci stupiamo in quanto $15 \,\%\, 12 = 3$.

Vediamo ora alcuni esempi di calcolo.

Esempio 21. Verificare che vale l'uguaglianza

$$(1235 + 765) \,\%\, 21 = ((1235 \,\%\, 21) + (765 \,\%\, 21)) \,\%\, 21$$

Infatti si ha

$$(1235 + 765) \,\%\, 21 = 2000 \,\%\, 21 = 5$$

inoltre

$$1235 \,\%\, 21 = 17, \qquad 765 \,\%\, 21 = 9$$

da cui anche $(17 + 9) \,\%\, 21 = 5$.

Esempio 22. Verificare che vale l'uguaglianza

$$(1235 \cdot 765) \,\%\, 21 = ((1235 \,\%\, 21) \cdot (765 \,\%\, 21)) \,\%\, 21$$

Infatti si ha

$$(1235 \cdot 765) \,\%\, 21 = 944775 \,\%\, 21 = 6$$

inoltre

$$1235 \,\%\, 21 = 17, \qquad 765 \,\%\, 21 = 9$$

da cui anche $(17 \cdot 9) \,\%\, 21 = 153 \,\%\, 21 = 6$.

2.1 Congruenze algebriche

Per tutte le applicazioni di crittografia che studieremo, risulta di importanza cruciale la definizione di *congruenza algebrica*[1].

Definizione 23. Diciamo che gli interi a e b sono *congruenti modulo n*, con $n > 0$, se vale l'uguaglianza $a \,\%\, n = b \,\%\, n$ cioè se la divisione di a e b per n dà lo stesso resto. Scriveremo in questo caso

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

Esempio 24. Verificare che $4965 \equiv 765 \pmod{21}$. Basta calcolare i resti delle divisioni 4965 : 21 e 765 : 21. In entrambi i casi il risultato è 9.

In alcuni casi può essere comodo verificare la congruenza modulo n tra due numeri ricorrendo a una definizione equivalente alla precedente.

Proposizione 25. *Condizione necessaria e sufficiente affinché sia* $a \equiv b \pmod{n}$ *è che* $a - b$ *sia un multiplo di* n.

Dimostrazione. Supponiamo che sia $a \equiv b \pmod{n}$ allora per la definizione di congruenza avremo

$$a = nx + r \qquad b = ny + r$$

da cui sottraendo membro a membro

$$a - b = nx - ny = n(x - y)$$

e quindi $(a - b)$ è un multiplo di n. Viceversa supponiamo che

$$a - b = nk$$

[1] Nelle Disquisitiones Arithmeticae (1801), K. F. Gauss mette in evidenza per la prima volta l'importanza della relazione di congruenza fra numeri interi (già nota e utilizzata da tempo) e della relativa aritmetica modulare. Per la relazione di congruenza Gauss fissa la notazione che utilizziamo tuttora e che la fa assomigliare all'uguaglianza.

ossia che $(a-b)$ sia un multiplo di n, allora $a = nk+b$. Inoltre, se dividiamo b per n avremo

$$b = nh + r$$

e sostituendo nell'uguaglianza precedente si ha

$$a = nk + nh + r = n(k+h) + r$$

da cui è evidente che $a \,\%\, n = b \,\%\, n = r$. □

2.2 Classi resto

La congruenza modulo n definisce una *relazione di equivalenza* nell'insieme dei numeri interi in quanto vale la seguente.

Proposizione 26. *Siano a, b e c numeri interi e sia $n > 0$ allora*

1. $a \equiv a \pmod{n}$
2. *se* $a \equiv b \pmod{n}$ *allora* $b \equiv a \pmod{n}$
3. *se* $a \equiv b \pmod{n}$ *e* $b \equiv c \pmod{n}$ *allora* $a \equiv c \pmod{n}$.

Dimostrazione. La prime due proprietà sono banali. Per verificare la terza osserviamo che per ipotesi deve essere

$$a - b = kn \qquad b - c = hn$$

dove k e h sono opportuni numeri interi. Sommando membro a membro si ottiene

$$a - c = kn + hn = (k+h)n$$

da cui si ha la tesi. □

Come noto una relazione di equivalenza permette di definire le relative *classi* che, nel nostro caso, saranno insiemi del tipo

$$[a]_n = \{b \in \mathbb{Z} : b \equiv a \pmod{n}\}.$$

Esempio 27. Sia $n = 4$ allora la classe $[1]_4$ coincide con l'insieme infinito

$$[1]_4 = \{\ldots, -7, -3, 1, 4, 7, \ldots\}.$$

È della massima importanza comprendere che esistono esattamente n *classi resto* che sono rispettivamente

$$[0]_n, [1]_n, \ldots, [n-1]_n.$$

Nel seguito indicheremo l'insieme delle classi resto modulo n con il simbolo $\mathbb{Z}_n$ e spesso indicheremo le classi semplicemente scrivendo il loro rappresentante. Così, ad esempio, anziché scrivere $[0]_n$ scriveremo semplicemente 0 e quindi scriveremo anche

$$\mathbb{Z}_n = \{0, 1, \ldots, n-1\}.$$

Nota 28. Osserviamo che quanto trattato fino a ora in questo paragrafo rispecchia fedelmente quanto accade quando si trattano i numeri razionali. La relazione di equivalenza è in questo caso la ben nota equivalenza tra frazioni

$$\frac{a}{b} \sim \frac{c}{d} \iff ad = bc$$

mentre le classi di equivalenza sono formate da tutte le frazioni equivalenti a una data. L'insieme delle classi di equivalenza è l'insieme dei numeri razionali:

$$\mathbb{Q} = \left\{\frac{1}{2}, -\frac{1}{2}, \ldots\right\}.$$

2.3 Aritmetica in $\mathbb{Z}_n$

Vediamo ora di definire alcune operazioni nell'insieme $\mathbb{Z}_n$.

2.3.1 Addizione

Siano $a, b \in \mathbb{Z}_n$ definiamo la loro *somma modulo n* semplicemente come

$$a + b \pmod{n} = (a + b) \,\%\, n.$$

Il problema matematico che si pone consiste nell'accertarsi che questa definizione si una *buona* definizione, ossia che non dipenda dai rappresentanti delle classi resto che sommiamo. In altri termini dobbiamo verificare che se $a \equiv b \pmod{n}$ e $c \equiv d \pmod{n}$ allora

$$a + c \equiv b + d \pmod{n}$$

La verifica è semplice in quanto se $a \equiv b \pmod{n}$ e $c \equiv d \pmod{n}$ allora

$$a - b = kn \qquad c - d = hn$$

da cui sommando membro a membro si ottiene

$$(a - b) + (c - d) = kn + hn \iff (a + c) - (b + d) = (h + k)n$$

e quindi $a + c \equiv b + d \pmod{n}$.

Nota 29. Si osservi che il controllo del fatto che la definizione di addizione modulo n sia buona costituisce l'analogo di ciò che accade quando viene definita l'addizione tra numeri razionali nel qual caso sappiamo che

$$\frac{a}{b} \sim \frac{a'}{b'} \quad \frac{c}{d} \sim \frac{c'}{d'} \Longrightarrow \left(\frac{a}{b} + \frac{c}{d}\right) \sim \left(\frac{a'}{b'} + \frac{c'}{d'}\right).$$

2.3.2 Moltiplicazione

Siano $a, b \in \mathbb{Z}_n$ definiamo il loro *prodotto modulo n* semplicemente come

$$ab \pmod{n} = (ab) \,\%\, n.$$

Analogamente a quanto visto per l'addizione, dobbiamo procedere alla verifica che questa definizione si una *buona* definizione ossia che non dipenda dai rappresentanti delle classi resto che moltiplichiamo. In altri termini dobbiamo verificare che se $a \equiv b \pmod{n}$ e $c \equiv d \pmod{n}$ allora

$$ac \equiv bd \pmod{n}.$$

La verifica è semplice in quanto se $a \equiv b \pmod{n}$ e $c \equiv d \pmod{n}$ allora

$$a - b = kn \qquad c - d = hn.$$

Ora osserviamo che

$$ac - bd = ac - bc + bc - db = ckn + bhn = (ck + bh)n$$

e quindi $ac \equiv bd \pmod{n}$.

Nota 30. Si osservi che il controllo del fatto che la definizione di moltiplicazione modulo n sia buona costituisce l'analogo di ciò che accade quando viene definita la moltiplicazione tra numeri razionali nel qual caso sappiamo che

$$\frac{a}{b} \sim \frac{a'}{b'} \quad \frac{c}{d} \sim \frac{c'}{d'} \Longrightarrow \left(\frac{a}{b} \cdot \frac{c}{d}\right) \sim \left(\frac{a'}{b'} \cdot \frac{c'}{d'}\right).$$

2.3.3 Criteri di divisibilità

Le operazioni che abbiamo appena definito nell'aritmetica modulare ci permettono di spiegare come sono stati stabiliti i *criteri di divisibilità* che conosciamo dalle scuole medie. Vediamo a titolo di esempio la deduzione del criterio di divisibilità per 3. Iniziamo con un esempio prendendo il numero 8754, osserviamo che esso è divisibile per 3 se e solo se $8754 \equiv 0 \pmod{3}$. Ora possiamo scrivere

$$8754 = 8 \cdot 10^3 + 7 \cdot 10^2 + 5 \cdot 10^1 + 4$$

da cui calcolando la riduzione modulo 3 di ciascuna potenza di 10 si ha:

$$8754 = 8 \cdot 10^3 + 7 \cdot 10^2 + 5 \cdot 10^1 + 4 \equiv (8 \cdot 1 + 7 \cdot 1 + 5 \cdot 1 + 4) \pmod 3$$

da cui possiamo stabilire che 8754 è divisibile per 3 solo se lo è il numero 24 che rappresenta la somma delle sue cifre. Essendo 24 un multiplo di 3 si ha $24 \equiv 0 \pmod 3$. Possiamo facilmente generalizzare quanto visto nell'esempio.

Proposizione 31. *Sia $n=a_n a_{n-1} \dots a_0$ un numero naturale, allora n è divisibile per 3 se e solo se $a_0 + a_1 + \dots + a_n$ è divisibile per 3.*

Dimostrazione. Osserviamo prima di tutto che se $k \geqslant 1$ allora $10^k \equiv 1 \pmod 3$ e quindi

$$n = a_n \cdot 10^n + \dots + a_1 \cdot 10 + a_0 \equiv (a_0 + a_1 + \dots + a_n) \pmod 3$$

da cui si ottiene immediatamente la tesi. □

In modo del tutto analogo si trova il criterio di divisibilità per 9. Si lascia come esercizio per il lettore la determinazione dei criteri di divisibilità per 7 e per 11.

2.3.4 Inverso

Il problema che ora ci poniamo consiste nello studiare l'inverso rispetto alla moltiplicazione nell'insieme $\mathbb{Z}_n$.

Definizione 32. Sia $a \in \mathbb{Z}_n$, con $a \neq 0$, diremo che $a^{-1} \in \mathbb{Z}_n$ è l'inverso di a rispetto alla moltiplicazione se

$$a \cdot a^{-1} \equiv 1 \pmod n.$$

Nel seguito indicheremo con $\mathbb{Z}_n^*$ l'insieme formato dagli elementi invertibili di $\mathbb{Z}_n$. Per fissare le idee analizziamo il caso particolare $\mathbb{Z}_5$ in cui è semplice e poco laborioso costruirsi la Tavola della moltiplicazione.

·	**0**	**1**	**2**	**3**	**4**
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

Tabella 2.1: Tavola della moltiplicazione in $\mathbb{Z}_5$

Non è difficile osservare che *ogni* elemento non nullo di $\mathbb{Z}_5$ ammette inverso. Infatti l'inverso di 1 è 1, di 2 è 3, di 3 è 2 e di 4 è 4. Quindi $\mathbb{Z}_5^* = \mathbb{Z}_5 - \{0\}$. Se ora

analizziamo il caso $\mathbb{Z}_6$ noteremo che la situazione è alquanto diversa. Infatti come possiamo notare dalla Tabella della moltiplicazione ci sono ben tre numeri (2, 3 e 4) che *non* ammettono inverso moltiplicativo! In questo caso $\mathbb{Z}_6^* = \{1,5\}$.

$\cdot$	**0**	**1**	**2**	**3**	**4**	**5**
0	0	0	0	0	0	0
1	0	1	2	3	4	5
2	0	2	4	0	2	4
3	0	3	0	3	0	3
4	0	4	2	0	4	2
5	0	5	4	3	2	1

Tabella 2.2: Tavola della moltiplicazione in $\mathbb{Z}_6$

La questione è chiarita in modo completo dalla proposizione seguente.

Proposizione 33. *Sia $a \in \mathbb{Z}_n$, allora a ammette inverso rispetto alla moltiplicazione se $(a,n) = 1$, ossia se a è coprimo con n.*

Dimostrazione. Supponiamo che sia $(a,n) = 1$ allora l'identità di Bézout ci informa del fatto che devono esistere due interi u e v tali che

$$1 = au + nv$$

da cui si ha $au \equiv 1 \pmod{n}$ e quindi u è l'inverso di a rispetto alla moltiplicazione. □

Corollario 34. *Se p è primo allora in $\mathbb{Z}_p$ ogni elemento $a \neq 0$ ammette inverso moltiplicativo.*

In altri termini, se p è primo i $p-1$ elementi non nulli di $\mathbb{Z}_p$ ammettono tutti un inverso rispetto alla moltiplicazione e quindi $\mathbb{Z}_p^* = \mathbb{Z}_p - \{0\}$ è quello che algebricamente si definisce un *campo*[2].

In generale, per calcolare l'inverso di a si può procedere seguendo quanto visto nella dimostrazione della Proposizione 33 ossia:

(a) con l'algoritmo di Euclide esteso si trova un elemento u tale che

$$1 = au + nv \implies au - 1 = -nv$$

(b) dalla relazione precedente si ha allora $au \equiv 1 \pmod{n}$ e conseguentemente $a^{-1} \equiv u \pmod{n}$.

Esempio 35. L'inverso di 5 in $\mathbb{Z}_9$ è 2. Infatti $1 = 5 \cdot 2 - 9 \cdot 1$ da cui $5 \cdot 2 \equiv 1 \pmod{9}$.

[2] Il campo maggiormente studiato nella scuola media è quello dei numeri razionali $\mathbb{Q}$.

Funzione di Eulero

Dato $n > 1$ introduciamo ora la funzione di Eulero[3] $\varphi(n)$ che per definizione ci indica il numero di elementi[4] di $\mathbb{Z}_n^*$. La funzione di Eulero gode delle seguenti proprietà:

(a) $\varphi(n) \leq n-1$ e in particolare $\varphi(n) = n-1$ se n è primo;

(b) se $n = pq$ allora $\varphi(n) = (p-1)(q-1)$;

(c) se $n = p^\alpha$, con $\alpha > 1$, allora $\varphi(n) = p^{\alpha-1}(p-1)$;

(d) se $n = p_1^{\alpha_1} \cdots p_n^{\alpha_n}$, con $\alpha_i > 1$ per $i = 1, \ldots, n$, allora

$$\varphi(n) = (p_1 - 1) \cdot p_1^{\alpha_1 - 1} \cdots (p_n - 1) \cdot p_n^{\alpha_n - 1}$$

2.4 Il Piccolo Teorema di Fermat

Le tecniche crittografiche che descriveremo nel prossimo capitolo sono conseguenza di un teorema noto come Piccolo Teorema di Fermat[5]. Prendiamo le mosse dal seguente importante.

Lemma 36. *Siano $x, y \in \mathbb{Z}_n$ con $x \neq y$. Dato $a \in \mathbb{Z}^*$ coprimo con n, ax e ay non sono mai tra loro congruenti modulo n.*

Dimostrazione. Supponiamo per assurdo che sia $ax \equiv ay \pmod{n}$ e, senza perdita di generalità, che sia anche $x > y$. Allora avremo

$$n \mid (x-y)a$$

da cui, ricordato che per ipotesi a è coprimo con n, discende che $n \mid (x-y)$. Ora $(x-y) \in \mathbb{Z}_n$ e $(x-y) \neq 0$ e quindi non può essere che $n \mid (x-y)$. □

Teorema 37 (Piccolo Teorema di Fermat). *Sia $p > 1$ un numero primo, allora per ogni numero $a \in \mathbb{Z}_p^*$ si ha*

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

[3] La funzione fu introdotta dal grande matematico svizzero L. Euler (1707-1783) nell'ambito dei suoi studi su alcuni teoremi di Fermat.

[4] Più precisamente dovremmo dire che la funzione di Eulero indica la cardinalità di $\mathbb{Z}_n^*$.

[5] Pierre de Fermat scoprì il teorema attorno al 1636. L'enunciato compare in una delle sue lettere, datata 18 ottobre 1640, al confidente Frenicle. La prima pubblicazione della dimostrazione del teorema è dovuta a L. Euler nel 1736 anche se una dimostrazione precedente è stata scoperta in alcuni manoscritti di Leibniz.

Dimostrazione. Consideriamo il sottoinsieme di $\mathbb{Z}_p$ formato dai multipli di a

$$A = \{a, 2a, 3a, \ldots, (p-1)a\}$$

che, per il Lemma 36, contiene $p-1$ elementi distinti diversi da 0. Perciò A coincide con il sottoinsieme B di $\mathbb{Z}_p$ definito da

$$B = \{1, 2, 3, \ldots, (p-1)\}.$$

Moltiplicando tra loro tutti gli elementi di A e tutti gli elementi di B otterremo allora lo stesso risultato

$$a \cdot 2a \cdot 3a \cdot \cdots \cdot (p-1)a \equiv 1 \cdot 2 \cdot 3 \cdot \cdots \cdot (p-1) \pmod{p}.$$

Ricordando che $2, 3, \ldots, (p-1)$ sono invertibili, possiamo semplificare ottenendo la tesi. □

Esempio 38. Sia $p = 5$ e calcoliamo a^4 per $a = 1, 2, 3, 4$.

$$1^4 = 1 \equiv 1 \pmod{5}, \qquad 2^4 = 16 \equiv 1 \pmod{5},$$
$$3^4 = 81 \equiv 1 \pmod{5}, \qquad 4^4 = 256 \equiv 1 \pmod{5}.$$

Nota 39. Il Teorema 37 *non* vale se p non è un numero primo. Un controesempio si ottiene nel caso particolare in cui sia $p = 6$ e $a = 2$ nel qual caso avremo $2^5 \equiv 2 \pmod 6$ e non $2^5 \equiv 1 \pmod 6$.

Corollario 40. *Sia p un numero primo e sia r un intero coprimo con $p-1$. Allora la funzione potenza $f(x) = x^r$ è invertibile in $\mathbb{Z}_p$ e la sua inversa è $f^{-1}(x) = x^s$ dove $rs \equiv 1 \pmod{p-1}$.*

Dimostrazione. Verifichiamo che $f^{-1}(f(x)) = x$ per ogni intero x, ossia che

$$x^{rs} = x \pmod{p}$$

per ogni intero x. Per ipotesi $rs \equiv 1 \pmod{p-1}$ e quindi esiste un intero b tale che

$$rs = 1 + b(p-1)$$

da cui segue che

$$x^{rs} = x^{1+b(p-1)} = x \cdot (x^{p-1})^b \equiv x \cdot 1 \pmod{p}$$

dove l'ultimo passaggio segue dal Piccolo Teorema di Fermat. □

Corollario 41. *Sia $n = pq$ con p e q numeri primi distinti e sia r un intero coprimo con $\varphi(n)$, allora la funzione potenza $f(x) = x^r$ è invertibile in $\mathbb{Z}_n$ con inversa la funzione $f^{-1}(x) = x^s$, dove $rs \equiv 1 \pmod{\varphi(n)}$.*

Dimostrazione. Verifichiamo che per ogni numero intero x vale la relazione $x^{rs} \equiv x \pmod{n}$. Ricordiamo che per le proprietà della funzione di Eulero viste in precedenza $\varphi(n) = (p-1)(q-1)$ e quindi che

$$rs \equiv 1 \pmod{\varphi(n)} \implies (p-1)(q-1) \mid (rs-1).$$

In particolare avremo che

$$(p-1) \mid (rs-1) \implies rs \equiv 1 \pmod{p-1}$$

e quindi, utilizzando il Corollario 40, anche

$$x^{rs} \equiv x \pmod{p}$$

con x intero. Dunque $p \mid (x^{rs} - x)$. Lo stesso ragionamento si applica a q e si conclude che allora $q \mid (x^{rs} - x)$. Siccome p e q sono due primi distinti, si conclude che

$$n = pq \mid (x^{rs} - x) \implies x^{rs} \equiv x \pmod{n}$$

per ogni intero x. □

2.5 Aritmetica modulare in Pari

Pari implementa un certo numero di operazioni modulari predefinite tra cui addizione e moltiplicazione. Le funzioni utili per eseguire calcoli modulo n sono `Mod(numero,n)` e `lift(numero)`.

Esempio 42. Calcolare in $\mathbb{Z}_{58}$: $(49+57) \pmod{58}$ e $49 \cdot 57 \pmod{58}$.

Se scriviamo `Mod(49+57,58)`, otterremo come risultato `48 mod58` mentre se si aggiunge il comando `lift` scrivendo `lift(Mod(49+57,58))` si ottiene direttamente `48`. L'azione di `lift` è quella di riportare il prodotto al suo valore in $\mathbb{Z}_{58}$. Analogamente si procede per il prodotto il cui risultato è `9`. Un possibile quesito che potrebbe sorgere spontaneo riguarda la differenza che sussiste tra `lift` e `Mod`. La risposta è che `Mod` (a differenza di `lift`) mantiene il riferimento alla modularità del numero, fatto che potrebbe risultare comodo quando si desidera ad esempio eseguire più operazioni modulari in sequenza.

2.5.1 Tavola della moltiplicazione

In alcuni casi può essere utile disporre della tavola della moltiplicazione modulo n una volta assegnato un dato valore n. Per poter disporre della tavola della moltiplicazione eseguiamo le seguenti azioni:

1. con un editor di testo qualsiasi apriamo un file che chiamiamo `prodmod.gp` e copiamo al suo interno il seguente testo:

```
/*******************************************************
*            TAVOLA DELLA MOLTIPLICAZIONE MODULO n
*   input: n - intero che definisce la tabella
*   output: matrice dei risultati delle moltiplicazioni
*            A. CENTOMO per il PLS 2005-2007
*******************************************************/

{Modprod(n) = local(a,i,j);

      a=matrix(n,n);

      for(i=1, n,

              for(j=1, n, a[i,j]=lift(Mod(i*j-i-j+1,n)));

         );

print("Tavola della moltiplicazione modulo ", n);

return(a);
}
```

2. una volta salvato il file scriviamo `\r prodmod` in modo che il file `prodmod.gp` venga letto dal programma;

3. quindi, se ad esempio vogliamo la tabella della moltiplicazione in $\mathbb{Z}_7$, scriviamo il comando `Modprod(7)`.

2.6 Complessità di un algoritmo

A dispetto della brevità di questo paragrafo i suoi contenuti sono indispensabili per poter comprendere le problematiche relative alla sicurezza nei sistemi di crittografia a chiave pubblica che studieremo nel prossimo capitolo.

Quando si parla di *complessità algoritmica* si fa riferimento al problema della classificazione degli algoritmi in base alle risorse computazionali (memoria occupata e tempo di calcolo) richieste per il loro funzionamento. Un approccio generale allo studio della complessità esula dagli scopi di questa introduzione elementare alla crittografia. Tuttavia vediamo innanzitutto come ricorrendo all'ambiente Pari sia possibile introdurre il problema.

Primalità e fattorizzazione

Dato un numero intero n è bene ricordare che sono da considerarsi *fatti distinti*:

(a) stabilire se n è un numero primo,

(b) determinare la scomposizione in primi di n.

Abbiamo visto che disponendo della scomposizione in fattori primi di un numero si può stabilire in modo ovvio se esso è o meno primo. Tuttavia esistono algoritmi che permettono di indagare la primalità di un numero senza dover ricorrere alla scomposizione in fattori primi.

Il problema che ora affrontiamo, in modo molto empirico, consiste nel confrontare in un caso specifico i tempi di esecuzione degli algoritmi di primalità e di fattorizzazione disponibili in PARI.

Esempio 43. Si consideri il numero

$$n = 175781558773882957827802596282126667866268825970404047$$

e si stabilisca se n è primo. In caso negativo fattorizzarlo calcolando i tempi computazionali per le diverse operazioni.

```
? n=175781558773882957827802596282126667866268825970404047
%1 = 175781558773882957827802596282126667866268825970404047

? isprime(n)
%2 = 0

? ##
*** last result computed in 0 ms.

? factor(n)
%3 =
[3 1]
[137 1]
[181 1]
[1523 1]
[1811 1]
[2729 1]
[3671 1]
[4957 1]
[5531 1]
[89681 1]
[96079 1]
[571789 1]
[633081103483 1]
```

```
[89681 1]
[96079 1]

? ##
*** last result computed in 9 ms.
```

Come si può vedere n non è primo e il tempo impiegato per la fattorizzazione è superiore a quello per stabilirne la primalità. Provando con diversi numeri primi (con un buon numero di cifre) quello che si ottiene è che il tempo impiegato dall'algoritmo di fattorizzazione per restituire la scomposizione in fattori di un numero composto risulta *sempre* maggiore (in alcuni casi drammaticamente maggiore) di quello impiegato dall'algoritmo di primalità per decidere se un numero è o meno primo. Da ciò siamo portati a sospettare che la complessità dell'algoritmo `isprime` sia notevolmente inferiore a quella di `factor`.

Importante. Il lettore interessato ad approfondire la questione potrebbe visualizzare il codice sorgente del *software libero* PARI per studiare la struttura delle funzioni implementate in `isprime` e in `factor` per poi procedere allo studio della loro complessità computazionale.

RSA Factoring Challenge

Per persuadere il lettore della difficoltà computazionale di scomporre grandi numeri primi in fattori ricordiamo l'esistenza dell'RSA Factoring Challenge, una sfida proposta da RSA Laboratories a partire dal Marzo del 1991 per incoraggiare la ricerca nel campo della teoria dei numeri computazionale. Gli RSA Laboratories pubblicarono[6] una lista di semiprimi – numeri che hanno esattamente due fattori primi – conosciuti come numeri RSA, con un premio in denaro per chi fosse riuscito a fattorizzarli. Il più piccolo di questi, un numero con 100 cifre decimali è chiamato RSA-100, fu fattorizzato in pochi giorni, ma molti dei numeri più grandi non sono stati ancora fattorizzati e ci si aspetta che rimarranno tali ancora per un tempo relativamente lungo. Possiamo allora trarre la seguente conclusione:

*La complessità degli algoritmi **fino a oggi** conosciuti è tale per cui la determinazione della scomposizione in fattori primi di un numero intero con un sufficiente numero cifre richiede tempi enormi anche per calcolatori molto potenti.*

Come vedremo, questo aspetto è il perno su cui si fonda la sicurezza della crittografia a chiave pubblica.

[6] La lista è disponibile al sito `http://www.rsa.com/rsalabs/node.asp?id=2093`.

Capitolo 3

Crittografia

La parola *crittografia* deriva etimologicamente dalle parole greche χρυπτός (nascosto) e γράφειν (scrivere) e quindi il suo significato letterale è quello di "scrivere nascosto". Più semplicemente possiamo descrivere l'oggetto della crittografia come segue:

> Alice desidera spedire a Bruno un messaggio segreto che purtroppo deve necessariamente passare per le mani di intermediari indesiderati. Per compiere questa operazione in sicurezza i due utilizzano la seguente strategia:
>
> 1. Alice cifra il messaggio in chiaro in modo da renderlo incomprensibile a estranei tramite un procedimento concordato da entrambi;
> 2. Bruno inverte il procedimento e decifra il messaggio.

3.1 Cifrario di Cesare

Vediamo subito un esempio di metodo crittografico che, stando alle testimonianza che Svetonio ci ha tramandato nelle *Vite dei Cesari*, era utilizzato da Giulio Cesare per le sue comunicazioni riservate. Supponiamo che l'alfabeto chiaro sia l'usuale alfabeto di 26 lettere.

→ Alfabeto chiaro: 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

Allora l'alfabeto cifrante nel metodo di Cesare è l'alfabeto ottenuto spostando di tre posti ciascuna lettera dell'alfabeto chiaro.

→ Alfabeto cifrante: 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

In questo modo il messaggio X alea iacta est[1] viene cifrato in A dohd ldfwd hvw che risulta indecifrabile – almeno a prima vista – da un intermediario che non sia a conoscenza del metodo impiegato per la cifratura. Questo metodo crittografico in verità non è affidabile in quanto risulta assai facilmente decrittabile; nessuno oggi si sognerebbe di utilizzarlo per comunicazioni ad alto grado di riservatezza. Tuttavia è istruttivo, soprattutto in vista di quanto vedremo nel seguito, affrontare il problema della riformulazione del metodo di Cesare nel linguaggio algebrico moderno. Per fare questo procediamo per passi.

(a) *Sostituiamo l'alfabeto con un insieme di numeri*: il modo più banale con cui effettuare questa sostituzione consiste nel sostituire ciascuna lettera con il numero che identifica il suo posto nell'ordine alfabetico. In questo modo l'alfabeto viene a coincidere con l'insieme numerico

$$\mathbb{Z}_{26} = \{0,1,2,\ldots,18,19,20,21,22,23,24,25\}$$

(b) *descriviamo il metodo di cifratura attraverso una funzione*

$$f\colon \mathbb{Z}_{26} \to \mathbb{Z}_{26}$$

che trasforma l'alfabeto chiaro nell'alfabeto cifrante. Dal momento che nel metodo di Cesare la cifratura avviene spostando di tre posti le lettere nell'ordine alfabetico una scelta naturale sembrerebbe data da

$$f(x) = x+3$$

In effetti, se prendiamo la lettera a dell'alfabeto chiaro, che corrisponde al numero 0, avremo che $f(0) = 0+3 = 3$ che corrisponde correttamente alla lettera d che occupa la quarta posizione nell'ordine alfabetico. Tuttavia i problemi nascono se prendiamo ad esempio la lettera x, che corrisponde al numero 24, in quanto $f(24) = 24+3 = 27 \notin \mathbb{Z}_{26}$. Per superare il problema possiamo ricorrere all'addizione modulare e scrivere

$$f_3(x) = x+3 \pmod{26}$$

da cui $f_3(24) = 1$.

(c) *Descriviamo il metodo di decifratura attraverso la funzione inversa di* f:

$$f^{-1}\colon \mathbb{Z}_{26} \to \mathbb{Z}_{26}$$

che nel caso del metodo di Cesare è

$$f_{23}(x) = x+23 \pmod{26}.$$

[1] La frase fu proferita da Cesare il 10 gennaio del 49 a.C. sulle rive del fiume Rubicone.

3.1.1 Implementazione in Pari

In questo paragrafo trattiamo un esempio di script più complesso di quelli analizzati in precedenza ma interessante per la crittografia in quanto implementa il metodo di Cesare. Lo script si fonda sull'uso di tre funzioni predefinite:

- → `Vecsmall`: trasforma una qualsiasi stringa di caratteri nei corrispondenti 256 numeri previsti dalla codifica ASCII[2];
- → `Strchr`: contrariamente alla precedente, trasforma numeri in caratteri;
- → `concat`: concatena i suoi argomenti.

Ad esempio:

```
? Vecsmall("crittografia")
%1 = Vecsmall([99,114,105,116,116,111,103,114,97,102,105,97])
? Strchr(%)
%2 = "crittografia"
```

In questo esempio la stringa `crittografia` è stata trasformata usando la funzione `Vecsmall` in una sequenza numerica e, successivamente, il risultato è stato ritrasformato nel testo di partenza utilizzando `Strchr`.

Per quanto concerne la funzione `concat`, per comprenderne il comportamento è sufficiente il seguente esempio:

```
? x=32
%1 = 32
? y="ciao"
%2 = "ciao"
? concat(x,y)
%3 = "32ciao"
```

dove il numero 32 viene concatenato alla stringa `ciao`.

A questo punto possiamo proporre il testo dello script che implementa il metodo di Cesare.

[2]Acronimo di American Standard Code for Information Interchange, è un sistema di codifica dei caratteri proposto dall'ingegnere B. Bemer nel 1961. Successivamente accettato come standard dall'ISO, è attualmente utilizzato nella versione a 8 bit per cui a ciascuno dei 256 caratteri in esso contenuti viene associata una sequenza di otto cifre binarie (0 o 1).

```
/*****************************************************
*            CRITTOSISTEMA DI CESARE
*         A. CENTOMO per il PLS 2005-2007
*****************************************************/

{alfabeto=["A","B","C","D","E","F","G","H","I","J", "K","L",\
"M","N","O","P","Q","R","S","T","U","V","W","X","Y","Z"];}

{da_lettera_a_numero(lettera)=local(j);
 for(j=1,26, if(alfabeto[j]==lettera,return(j)));
 error("input non valido.")
}

{
 CIFRA(messaggio, chiave="", codifica="") = local(l,i,j,k,C,M);

 l=length(messaggio);
 M=Vecsmall(messaggio);

for(j=1,l, C = da_lettera_a_numero(Strchr(M[j])) + 3;
 if( C % 26 == 0, C = 26, C = C % 26);
 codifica = concat(codifica,alfabeto[C]);
   );
print("Il messaggio codificato e'");
print(codifica);
return(codifica);
}
{
 DECIFRA(codifica, messaggio="") = local(l,i,j,k,M,C);

 l=length(codifica);
 C=Vecsmall(codifica);

 for(j=1,l, M=da_lettera_a_numero(Strchr(C[j])) - 3;
  if (M % 26 == 0, M = 21, M = M % 26);
  messaggio = concat(messaggio,alfabeto[M])
    );
 print("Il messaggio decodificato e'");
 print(messaggio);
 return(messaggio); }
```

3.1.2 Debolezza del metodo di Cesare

Per porre in luce la debolezza del metodo di Cesare vediamo come esso possa essere attaccato con successo utilizzando l'analisi di frequenza. Consideriamo il seguente messaggio cifrato che Giulia ha scoperto tra le pagine delle *Vite dei Cesari* di Svetonio del fidanzato, scritto su un foglio che recava la scritta "A Lisa". Giulia sospetta che ci sia del tenero: avrà buone ragioni per pensare questo?

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

Si tratta di un messaggio di 152 caratteri distribuiti come segue:

V	23	E	6	A	4
P	20	S	5	W	2
Y	14	N	5	M	2
L	14	B	5	J	2
U	11	K	5	I	1
T	11	C	5	O	1
H	11	Z	4	X	1

Giulia legge nel testo di Svetonio la tecnica di cifratura di Cesare e prova a decrittare il messaggio spostando di tre posti le lettere dell'alfabeto ma la cosa non funziona. Allora ipotizza che sia stata usata la stessa tecnica ma con una chiave diversa da 3.

Lettera	A	B	C	D	E	F	G	H	I	L	M
%	2.5	11.7	0.9	4.5	3.7	11.7	0.9	1.6	1.5	11.3	6.5
Lettera	N	O	P	Q	R	S	T	U	V	Z	
%	6.9	9.8	3.0	0.5	6.4	5.0	5.6	3.0	2.1	0.5	

Tabella 3.1: Frequenze lingua italiana tratte da [9]

Sapendo che Lisa non conosce il latino e noto che nella lingua italiana la distribuzione di frequenza delle lettere è rappresentata nella Tabella 3.1, Giulia cerca la chiave di cifratura n osservando i seguenti fatti.

Le lettere più frequenti nella lingua italiana sono le quattro vocali A, E, I e O mentre nel testo cifrato sono le lettere V, P, Y e L.

- Prima ipotesi: A → V, allora $n = 21$. In questo caso tuttavia Q → L e ciò è assai poco probabile in quanto la lettera L è molto frequente nel testo cifrato contrariamente alla lettera Q nella lingua italiana.

- Seconda ipotesi: E → V, allora $n = 17$. In questo caso Y → P e la cosa appare molto improbabile.
- Terza ipotesi: I → V, allora $n = 13$. In questo caso Y → L e la cosa appare molto improbabile.
- Quarta ipotesi: O → V allora $n = 7$. In questo caso P → I, Y → R e L → E; quindi potrebbe essere che si sia in presenza della chiave giusta. La chiave 14 dovrebbe allora permettere di decrittare il messaggio.

In effetti si tratta di un adattamento abbastanza compromettente del Carme V del poeta latino Valerio Catullo.

> Viviamo mia Lisa e amiamo e ogni mormorio perfido dei vecchi valga per noi un soldo il giorno muore e risorge ma quando muore il nostro breve giorno una notte infinita dormiremo x x x x x

Non è difficile immaginare che anche senza conoscere il metodo con cui è stato cifrato il messaggio, attraverso tecniche tipo l'analisi di frequenza di lettere, digrafi e trigrafi si sarebbe venuti a capo del testo chiaro in non molto tempo. Da questo punto di vista il metodo di Cesare si rivela un metodo crittografico debole. Altri esempi di decrittazione interessanti e tratti dalla letteratura si trovano in [11].

3.2 Il codice di Vigenère

Un codice crittografico che per molto tempo è stato considerato affidabile è dovuto a B. de Vigenère (1523 – 1596) e consiste nell'utilizzo simultaneo di più funzioni di cifratura di Cesare. Chiariamo il metodo attraverso la discussione di un esempio. Supponiamo di lavorare con lo stesso alfabeto di 26 lettere utilizzato per la descrizione del cifrario di Cesare e di voler cifrare il seguente testo[3] usando il codice di Vigenère.

> sono hormai trenta anni che per haver spie mi truovo qui

Scegliamo innanzitutto una chiave segreta, ad esempio la parola sole, e quindi riscriviamo il testo in chiaro in blocchi che contengono un numero di lettere pari a quello della chiave ossia in blocchi di quattro lettere come nella parte sinistra della Tabella 3.2.

Nell'alfabeto di 26 lettere che abbiamo scelto la chiave sole corrisponde alla quaterna di numeri $(18, 14, 11, 4)$ (s è la lettera 18, o la lettera 14, eccetera). Consideriamo allora la quaterna di funzioni di cifratura di Cesare

$$(f_{18}, f_{14}, f_{11}, f_4)$$

[3] Tratto dalla lettera scritta il 9 Dicembre del 1563 dall'agente segreto spagnolo B. Prototico e indirizzata al re di Spagna.

s	o	l	e
s	o	n	o
h	o	r	m
a	i	t	r
e	n	t	a
a	n	n	i
c	h	e	p
e	r	h	a
v	e	r	s
p	i	e	m
i	t	r	u
o	v	o	q
u	i	x	y

f_{18}	f_{14}	f_{11}	f_4
k	c	y	s
z	c	c	q
s	w	e	v
w	b	e	e
s	b	y	m
u	v	p	t
w	f	s	e
n	s	c	w
h	w	p	q
a	h	c	y
g	j	z	u
m	w	i	c

Tabella 3.2: Codice di Vigenère: testo chiaro e testo cifrato

e utilizziamo ciascuna di esse per cifrare rispettivamente la prima, la seconda, la terza e la quarta colonna del messaggio a blocchi della Tabella 3.2.

La sostituzione polialfabetica operata dal codice di Vigenère rende più complessa la decrittazione dei messaggi ma il metodo presenta delle debolezze che via via hanno condotto al suo superamento. Il primo a pubblicare un metodo di decrittazione del codice di Vigenère fu il colonnello prussiano F. Kasiski nel 1863.

3.3 Crittografia a chiave pubblica

In generale chiameremo *crittosistema* una qualsiasi terna del tipo

$$(\mathbb{Z}_n, f, f^{-1})$$

dove $\mathbb{Z}_n$ rappresenta un alfabeto di n simboli, f è una funzione di cifratura e f^{-1} (inversa di f) è la funzione di decifratura. Inoltre diremo in modo informale che un crittosistema è a *chiave pubblica* se dalla conoscenza di f non è possibile ricavare f^{-1} senza avere a disposizione ulteriori informazioni. Chiaramente il crittosistema di Cesare non è a chiave pubblica in quanto una volta nota la funzione di cifratura è immediato ricavare quella di decifratura.

3.3.1 Il crittosistema RSA

L'algoritmo RSA è stato descritto per la prima volta nel 1977 da R. Rivest, A. Shamir e L. Adleman al MIT; le lettere RSA vengono proprio dalle iniziali dei cognomi. C. Cocks, un matematico britannico che lavorava per un dipartimento di spionaggio, il GCHQ,

descrisse un sistema equivalente in un documento interno nel 1973. I documenti furono posti sotto segreto e a quel tempo, visto il costo relativamente alto delle macchine necessario per implementarlo, non ci furono ulteriori indagini né prove pratiche e la cosa fu considerata come una curiosità, per quanto se ne sa. La scoperta di Cocks fu resa pubblica solo nel 1997. Nel 1983 l'algoritmo fu brevettato negli Stati Uniti dal MIT (brevetto 4.405.829). Il brevetto è scaduto il 21 settembre 2000.

Nel crittosistema RSA ciascun utente compie le seguenti azioni:

1. sceglie casualmente due numeri primi p e q distinti e grandi (circa trecento cifre) e calcola $n = pq$;
2. calcola la funzione di Eulero $\varphi(n) = (p-1)(q-1)$;
3. sceglie casualmente un intero r tale che $1 < r < \varphi(n)$ e *coprimo* con φn ossia tale che $(r, \varphi(n)) = 1$;
4. calcola $s \equiv r^{-1} \pmod{\varphi(n)}$ (cosa semplice per chi conosce sia p che q)
5. rende pubblica la coppia di numeri $K_E = (n, r)$, detta *chiave pubblica*, e tiene segreto il numero $K_D = s$, detto *chiave privata*.

La cifratura e la decifratura dei messaggi, che devono essere preventivamente trasformati in numeri, avviene attraverso le funzioni potenza definite nel Corollario 41:

- la funzione di cifratura è $f\colon \mathbb{Z}_n \to \mathbb{Z}_n$, $x \mapsto x^r \pmod{n}$ ed è completamente definita dalla chiave pubblica K_E;
- la funzione di decifratura è $f^{-1}\colon \mathbb{Z}_n \to \mathbb{Z}_n$, $x \mapsto x^s \pmod{n}$ ed è completamente definita dalla chiave privata K_D;

Il metodo funziona perché, sempre dal Corollario 41, sappiamo che $f^{-1}(f(x)) = x$, per ogni $x \in \mathbb{Z}_n$.

Nota 44. Naturalmente il messaggio, nella sua traduzione numerica x, deve risultare un elemento di $\mathbb{Z}_n$ e quindi deve soddisfare la condizione matematica $x < n$. Nelle implementazioni professionali del crittosistema RSA il valore di n è talmente elevato che il problema sostanzialmente non si pone. Tuttavia nei casi che tratteremo a titolo di esempio nel seguito vedremo che i messaggi cifrabili dovranno avere un numero di cifre non superiore ad alcuni valori.

Per chiarire ulteriormente il funzionamento del crittosistema RSA analizziamo una situazione pratica in cui sono coinvolti solo due utenti:

	Alice	Bruno
Chiavi Pubbliche	$K_{E_A} = (n_A, r_A)$	$K_{E_B} = (n_B, r_B)$
Chiavi Private	$K_{D_A} = s_A$	$K_{D_B} = s_B$

Alice vuole mandare a Bruno il messaggio segreto $m \in \mathbb{Z}_{n_B}$ e quindi usa la chiave pubblica di Bruno per calcolare il numero

$$c \equiv m^{r_B} \pmod{n_B}.$$

Bruno riceve c e applicando la sua funzione di decifratura calcola

$$c^{s_B} = m^{r_B s_B} \pmod{n_B} = m$$

e quindi è in grado di leggere il messaggio inviato da Alice.

Esempio 45. Supponiamo che Bruno abbia utilizzato: $p_B = 5$ e $q_B = 17$ da cui $n_B = 85$ e $\varphi(n_B) = 64$. Osservato che 21 è coprimo con 64 supponiamo che egli abbia scelto come chiave pubblica $K_{E_B} = (85, 21)$ e quindi che la sua chiave privata sia $K_{D_B} = 61$, in quanto $61 \cdot 21 \equiv 1 \pmod{64}$. Supponiamo che Alice gli abbia spedito un messaggio che tradotto in numero sia $m = 7$ allora il messaggio cifrato sarà

$$c = 7^{61} \pmod{85} = 62$$

Per decifrarlo Bruno deve calcolare

$$62^{61} \pmod{85} = 7$$

che coincide esattamente con m.

L'Esempio 45 ci permette di affrontare il passo successivo che consiste nel comprendere dove risieda la *sicurezza* del crittosistema RSA. Un eventuale spia che venisse in possesso del messaggio cifrato 62 per risalire al messaggio originale 7 dovrebbe tentare di indovinare la chiave privata di Bruno. Il compito sarebbe in questo caso piuttosto semplice in quanto la spia non troverebbe difficile determinare la scomposizione in primi $85 = 5 \cdot 17$ da cui è possibile risalire a $\varphi(n)$ e quindi alla chiave privata.

Tuttavia immaginiamo che i numeri primi p_B e q_B siano stati scelti *molto grandi* (circa trecento cifre) e *diversi*[4]. Il compito della spia diverrebbe immediatamente arduo in quanto per determinare $\varphi(n_B)$ si tratterebbe di risolvere il problema computazionalmente complesso che consiste nel risalire dal numero primo n_B alla sua fattorizzazione.

3.3.2 RSA in Pari

Vediamo ora di ripercorrere il crittosistema RSA utilizzando PARI in modo da utilizzare numeri più vicini a quelli dell'implementazione reale del metodo e per approfondire la trasformazione di un testo in numero.

[4] Esistono algoritmi veloci di fattorizzazione di numeri primi che sono ottenuti moltiplicando tra loro numeri primi molto vicini tra loro. Quindi la sicurezza del sistema RSA è legata alla scelta di primi "lontani" tra loro.

- *Generazione di p e q*

```
? p=nextprime(random(10^40))
%1 = 725682550234587355002600325177940059756 1
? isprime(p)
%2 = 1
? q=nextprime(random(10^35))
%3 = 79214481285059248805111509508972537
? isprime(q)
%4 = 1
```

La funzione `nextprime(x)`, che determina il più piccolo pseudoprimo maggiore o uguale a `x`, viene utilizzata insieme alla funzione `random(k)` che genera un numero casuale compreso tra 0 e $k-1$, per generare i primi p e q. La funzione predefinita `isprime` che esegue un test di primalità sul suo argomento viene utilizzata per confermare che p e q siano effettivamente primi.

- *Calcolo di n*

```
? n=p*q
%5 = 57484566794451788216542806860441459
250457625658944073537424971602893818225 7
```

- *Calcolo di $\varphi(n)$*

```
? phin=(p-1)*(q-1)
%6 = 57484566794451788216542806860441458
524767153976228212609944135274002861216 0
```

Osserviamo che il calcolo di n e del corrispondente valore $\varphi(n)$ della funzione di Eulero è computazionalmente agevole noti i numeri p e q.

- *Calcolo di r*

```
r=random(n)
%7 = 47000017476545391300694041761909628
611305051765001294349203844294288249847 3
? while(gcd(phin,r)!=1,r=r+1)
? r
%8 = 47000017476545391300694041761909628
611305051765001294349203844294288249847 3
```

Il calcolo del numero r che completa la chiave pubblica avviene in due fasi: prima si genera un numero casuale e quindi si cerca un suo valore successivo che sia coprimo con $\varphi(n)$.

- *Calcolo di* $s \equiv r^{-1} \pmod{\varphi(n)}$

```
? s = lift(Mod(r,phin)^(-1));
? (r*s)%phin
%10 = 1
? s
%11 = 348167157329844556295500852717108111
19551727134699800742412302186794482098 17
```

Il calcolo della chiave privata s è accompagnato dalla verifica del fatto che in effetti $sr \equiv 1 \pmod{\varphi(n)}$.

- *Calcolo della lunghezza massima di testo codificabile*

```
? log(n)/log(30)
%12 = 50.611654949776177874130565 28
```

Usando un alfabeto di 30 caratteri possiamo codificare in un blocco unico un testo avente meno di 50 caratteri. Infatti se x è la traduzione in numero di un messaggio, il numero di caratteri del messaggio corrisponde al numero di cifre di x in base 30 che è $\log_{30} x$. Ora il numero di cifre massimo ammissibile sarà pari al valore della parte intera di $\log_{30} n = \log n / \log 30$, dove si è utilizzata la formula del cambiamento di base dei logaritmi.

- *Codifichiamo* "GUARDALA":
- *Calcolo dell'equivalente numerico in base 30 di GUARDALA usando l'alfabeto*: A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, ;, „ ., '.

```
? m=6*30^7+20*30^6+0*30^5+17*30^4+3*30^3+0*30^2+11*30^1+0*30^0
%13 = 145813851330
```

- *Definizione della funzione di cifratura*

```
E(x)=lift(Mod(x,n)^r)
```

- *Definizione della funzione di decifratura*

```
D(x)=lift(Mod(x,n)^s)
```

- *Codifica di GUARDALA*

```
? secret = E(m)

%49 = 130703952309790486520081123002 8827
47588584100681614510924875291337261771685
```

```
? D(secret)

%50 = 145813851330
```

La procedura è stata sviluppata correttamente poiché l'equivalente numerico della decodifica è uguale a quello di GUARDALA.

3.3.3 Il crittosistema ElGamal

L'algoritmo ElGamal, proposto per la prima volta da T. Elgamal nel 1984, rappresenta uno dei crittosistemi a chiave pubblica maggiormente diffusi. Una modifica di questo sistema nota come DSA (Digital Signature Algorithm) è stata adottata dal governo degli Stati Uniti come sistema standard per la firma digitale.

Nel crittosistema ElGamal ciascun utente compie le seguenti azioni:

1. l'utente A sceglie un primo p con molte cifre e un numero g che assumiamo $1 < g < p$
2. sempre A sceglie casualmente un numero a con $1 < a < p$ e calcola la potenza $g^a \pmod{p}$
3. la chiave pubblica di A è la terna $K_{E_A} = (p, g, g^a)$ mentre la chiave privata è il numero $K_{D_A} = a$

Se l'utente B vuole mandare un messaggio M all'utente A procede come segue:

(a) controlla che sia $M < p$ in modo che $M \in \mathbb{Z}_p$;

(b) calcola, utilizzando tra le altre cose la sua chiave privata $K_{D_B} = b$, la quantità $M \cdot (g^a)^b \pmod{p}$ che poi spedisce

L'utente A, che riceve il messaggio cifrato, può procedere alla decifratura in due passi

I. conoscendo la chiave pubblica di B calcola $(g^b)^a \pmod{p}$

II. quindi calcola $M \cdot (g^a)^b \cdot ((g^b)^a)^{-1} \pmod{p} = M$.

La sicurezza del metodo è legata alla presunta notevole complessità computazionale del calcolo del *logaritmo discreto* che permette di risalire da un numero intero y all'esponente x tale che $y = g^x$.

Importante. Osserviamo che per i numeri reali la determinazione di x è immediata in quanto $x = \log_g y$. Tuttavia il calcolo di un'approssimazione del logaritmo di un numero reale positivo sfrutta proprietà quali la continuità, la derivabilità e via di seguito, che non hanno un equivalente nell'insieme dei numeri interi. In conclusione il calcolo del logaritmo discreto è un problema di natura essenzialmente diversa rispetto al suo analogo nell'insieme dei numeri reali.

Esempio 46. Alice sceglie $p = 11$, $g = 3$ e $a = 7$ e, per completare la sua chiave pubblica, calcola $g^a = 3^7 \pmod{11} = 9$. Quindi $K_{E_A} = (11,3,9)$.

Bruno vuole spedire ad Alice il messaggio $M = 5$ e dal momento che la sua chiave privata è $K_{D_B} = 8$ egli calcola $3^8 \pmod{11} = 5$. Quindi egli calcola

$$9^8 \pmod{11} = 3 \qquad 5 \cdot 3 \pmod{11} = 4$$

e spedisce $(5,4)$.

Alice per decifrare il messaggio calcola

$$5^7 \pmod{11} = 3 \qquad 4 \cdot 3^{-1} \pmod{11} = 16 \pmod{11} = 5 \pmod{11}$$

e scopre che $M = 5$.

3.3.4 ElGamal in Pari

Vediamo ora di ripercorrere il crittosistema ElGamal utilizzando PARI in modo da utilizzare numeri più vicini a quelli dell'implementazione reale del metodo.

- *Generazione di p*

```
? p=nextprime(random(10^40))
%1 = 725682550234587355002600325177940059756 1
? isprime(p)
%2 = 1
```

- *Generazione di g e della chiave privata* $K_{D_A} = a$

```
? g=random(p)
%3 = 589321771004188496444471787378561875685 3
? a=random(p)
%4 = 287219791437834072400823636769031917558 5
```

- *Calcolo di* g^a *e quindi* $K_{E_A} = (p, g, g^a)$

```
? ga=lift(Mod(g,p)^a)
%5 = 475274121245929692488076344260832144758 2
```

- *Generazione della chiave privata* $K_{D_B} = b$ *e di* g^b, $K_{E_B} = (p, g, g^b)$

```
? b=random(p)
%6 = 161925817106182236000133198274841222741 0
? gb=lift(Mod(g,p)^b)
%7 = 272550225782364680351988635409956464346 8
```

- *Calcolo della lunghezza di testo massimo codificabile*

```
? log(p)/log(30)
%8 = 26.985426249062628344324216 72
```

Usando un alfabeto di 30 caratteri possiamo codificare in un blocco unico un testo avente meno di 26 caratteri.

- *Come nell'implementazione di* RSA *supponiamo che Bruno spedisca il messaggio* "GUARDALA"

```
? M=145813851330
%9 = 145813851330
```

- *Funzione di cifratura*

```
? E(x)=lift(x*Mod(gb,p)^a)
```

- *Funzione di decifratura*

```
? D(x)=lift(x*(Mod(ga,p)^b)^-1)
```

- *Cifratura di* "GUARDALA"

```
? E(M)
%10 = 77599131118436117535680498973119090999 8
```

- *Decifratura*

```
 D(E(M))
%11 = 145813851330
```

3.4 Firma digitale

In questo paragrafo discutiamo uno schema generale di *firma digitale* ossia un sistema di autenticazione[5] di documenti analogo alla firma autografa. La firma digitale è l'equivalente elettronico di una tradizionale firma apposta su carta ed è associata stabilmente a un documento informatico che assume pertanto le seguenti caratteristiche:

[5] La firma digitale ha caratteristiche che superano la semplice sottoscrizione di un testo e rende possibile firmare con valore legale un documento informatico. Nell'ordinamento giuridico italiano la firma digitale a crittografia asimmetrica è riconosciuta ed equiparata a tutti gli effetti di legge alla firma autografa su carta.

Il primo atto normativo che ha stabilito la validità della firma digitale per la sottoscrizione dei documenti elettronici è stato il DPR 513 del 1997, emanato in attuazione dell'articolo 15 della legge 15 marzo 1997, n. 59. Successivamente, tale normativa è stata trasposta nel DPR n. 445 del 2000 (il Testo Unico sulla documentazione amministrativa), più volte modificato negli anni successivi all'emanazione, per conformare la disciplina italiana alla normativa comunitaria contenuta nella Direttiva 99/93 in materia di firme elettroniche. Oggi, la legge che disciplina la firma digitale è il decreto legislativo 7 marzo 2005, n. 82, recante "Codice dell'amministrazione digitale" così come modificato dal D. Lgs. 4 aprile 2006, n. 159.

- integrità: garanzia che il documento non sia stato manomesso dopo la sottoscrizione;
- autenticità: garanzia dell'identità di chi firma;
- non ripudio: l'autore non può disconoscere il documento firmato;
- valore legale: il documento elettronico sottoscritto digitalmente ha lo stesso valore legale di un documento cartaceo sottoscritto con firma autografa.

Supponiamo che utilizzando un crittosistema a chiave pubblica qualsiasi Alice e Bruno dispongano delle seguenti chiavi:

- Alice: f_A (chiave pubblica) e f_A^{-1} (chiave privata)
- Bruno: f_B (chiave pubblica) e f_B^{-1} (chiave privata)

Supponiamo che Alice abbia come nome convenzionale un numero s_A (potrebbe essere l'indirizzo ip del computer da cui spedisce i messaggi, il tempo di spedizione o altro). Chiamiamo *firma digitale* di Alice il numero

$$f_A^{-1}(s_A)$$

Se ora Alice vuole mandare a Bruno un messaggio M accompagnato dalla sua firma digitale spedirà la coppia

$$(f_B(M), f_B(f_A^{-1}(s_A))$$

Quando Bruno riceve il messaggio firmato è in grado, attarverso la sua chiave privata, sia di decifrare il messaggio che di verificare la firma di Alice:

$$f_B^{-1}(f_B(M)) = M \qquad f_A(f_B^{-1}(f_B(f_A^{-1}(s_A)))) = s_A$$

dove osserviamo che nell'ultimo passaggio è stata usata anche la chiave pubblica di Alice.

Esempio 47. Vediamo un esempio numerico concreto di firma digitale tramite chiavi RSA. Supponiamo i seguenti dati.

→ Alice: chiave pubblica $K_{E_A} = (319, 47)$ e privata $K_{D_A} = 143$;

→ Bruno: chiave pubblica $K_{E_B} = (253, 81)$ e privata $K_{D_A} = 201$;

Supponiamo che il nome convenzionale di Alice sia $s_A = 17$ allora la sua firma digitale sarà

$$f_A^{-1}(17) = 17^{143} \pmod{319} = 128.$$

Se il messaggio firmato da spedire a Bruno fosse $M = 11$ avremo

$$(f_B(10), f_B(128)) = (10^{81} \pmod{253}, 128^{81} \pmod{253}) = (120, 18).$$

Importante. Chiaramente solo Alice può calcolare con la sua chiave privata $f_A^{-1}(s_A)$. Tuttavia se un terzo utente intercettasse la firma digitale di Alice potrebbe spacciarsi per lei in un messaggio successivo utilizzando per la firma $f_A^{-1}(s_A)$.

Per evitare problemi di intercettazione una firma digitale si ottiene in generale dalla *combinazione tra chiave privata e testo del messaggio.* Usando la chiave pubblica del mittente si può verificare un messaggio e controllare non solo che il messaggio sia stato inviato dalla persona corretta, ma anche che il suo contenuto non sia stato modificato durante il trasporto. Prima di chiarire questi aspetti ricordiamo che per la firma digitale si ricorre tipicamente all'uso di *funzioni hash.* Una funzione hash

$$h\colon \mathbb{Z}_n \to \mathbb{Z}_n$$

non cifra il messaggio M ma lo trasforma in un messaggio più breve $h(M)$ detto *impronta* di M, in modo da renderlo irriconoscibile a chiunque (mittente compreso). Inoltre le funzioni hash godono della proprietà per cui dati due messaggi distinti M e M' la probabilità che essi abbiano le stessa impronta è molto piccola.

La *firma digitale* di un messaggio M è data quindi da $f_A^{-1}(h(M))$, ossia coincide con il messaggio ottenuto dal mittente (Alice) dopo aver decifrato con la sua chiave privata l'impronta $h(M)$. Bruno, il destinatario del messaggio M, avendo a disposizione la chiave pubblica di Alice, può calcolare

$$f_A(f_A^{-1}(h(M)) = h(M)$$

e confrontarlo con $h(M)$ calcolato utilizzando la funzione pubblica h. Il confronto ha esito negativo solo se il messaggio M è stato modificato oppure se la firma non è stata creata utilizzando la chiave privata di Alice:

- nel primo caso infatti Bruno riceve il messaggio $M' \neq M$ e quindi $h(M') \neq f_A(f_A^{-1}(h(M)) = h(M)$;
- nel secondo caso la firma sarebbe $f^{-1}(h(M)) \neq f_A^{-1}(h(M))$ e quindi si avrà anche la disuguaglianza $h(M) \neq f_A(f^{-1}(h(M))$.

3.4.1 Certificazione delle chiavi

Un problema che si pone nel momento in cui si utilizza la firma digitale consiste nell'assicurarsi della corrispondenza tra chiavi di cifratura e utente. Un utente C potrebbe dichiarare di essere l'utente B e distribuire la propria chiave pubblica f sotto falso nome. L'utente A per mandare un messaggio a B usa la chiave f e quindi C riceve un messaggio destinato a B.

Per ovviare a questo problema si introduce un *Ente Certificatore* delle chiavi pubbliche. Generalmente si tratta di una società super-partes che raccoglie e distribuisce le chiavi pubbliche degli utenti, partendo dal presupposto che gli utenti

nutrano piena fiducia nell'Ente. Così l'utente B comunica la sua chiave all'Ente Certificatore e se A deve comunicare con B chiede direttamente la chiave all'Ente che è in grado di certificare con sicurezza la proprietà della chiave. L'elenco pubblico dei certificatori di firma digitale è disponibile nel sito del CNIPA (Centro Nazionale per l'Informatica nella Pubblica Amministrazione)[6].

3.4.2 Marcatura temporale

La disponibilità di Enti certificatori per le chiavi pubbliche per i sistemi di firma digitale consente di risolvere il problema della *marcatura temporale* ossia della datazione precisa di un messaggio.

Supponiamo che A desideri spedire un messaggio M a B volendo essere sicuro che la data del messaggio non possa essere messa in discussione. Egli può procedere come segue:

- A invia all'Ente certificatore la quantità $f_B f_A^{-1}(h(M))$
- l'Ente aggiunge data e ora T al messaggio spedendo ad A un nuovo messaggio che contiene la quantità $f_A f_E^{-1}(f_B f_A^{-1}(h(M)), T)$
- l'utente A decodifica e ottiene $f_E^{-1}(f_B f_A^{-1}(h(M)), T)$
- A invia a B la quantità $f_B f_E^{-1}(f_B f_A^{-1}(h(M)), T)$
- B decodifica applicando $f_E f_B^{-1} f_B f_E^{-1}(f_B f_A^{-1}(h(M)), T)$ e quindi ottiene

  $$f_B f_A^{-1}(h(M)), T)$$

  da cui riconosce la firma di A e può risalire alla data T.

[6] Si consulti l'area Attività al sito `http://www.cnipa.gov.it/site/it-it/`.

Capitolo 4

Software per la Crittografia

GNU Privacy Guard (GNUPG o GPG), rilasciato sotto la licenza GPL, è un programma progettato per sostituire la suite crittografica PGP[1], originariamente sviluppata da P. Zimmermann nel 1991. GPG è completamente compatibile con gli standard Open PGP dell'IETF ed è sostenuto dal governo tedesco. Fa parte del software sviluppato dalla Free Software Foundation. In questo capitolo, che è stato redatto riferendosi a [4], si trova una descrizione dei principali comandi dell'ambiente GPG corradata da un buon numero di esempi pratici.

4.1 Gnu PG

Come abbiamo visto diffusamente in precedenza, la crittografia a chiave pubblica prevede due chiavi: una chiave pubblica, che può essere diffusa con ogni mezzo, e una chiave privata, che non va diffusa e deve essere mantenuta segreta dal proprietario. Più precisamente, indicati con $\mathscr{P}$ l'insieme dei possibili messaggi spedibili e con $\mathscr{C}$ l'insieme dei corrispondenti messaggi cifrati, una *chiave pubblica* è una qualsiasi procedura

$$f : \mathscr{P} \to \mathscr{C}$$

che a ogni messaggio associa il corrispondente messaggio cifrato. Viceversa la *chiave privata* associata alla chiave pubblica f è la procedura

$$f^{-1} : \mathscr{C} \to \mathscr{P}$$

che dal messaggio cifrato $f(M)$ permette di risalire a M, cioè la funzione tale che $f^{-1}(f(M)) = M$, dove M è un qualsiasi messaggio. In termini matematici possiamo

[1] Il lettore interessato all'avvincente storia della nascita di PGP è rinviato al capitolo *Una riservatezza niente male* del testo citato di S. Singh [9].

dire che f è una funzione da $\mathscr{P}$ a $\mathscr{C}$ e f^{-1} è la sua inversa. Lo scambio di messaggi avviene come segue: il mittente cifra il messaggio M con la chiave pubblica f_B del destinatario; questi riceve il messaggio cifrato $f_B(M)$ che decifra utilizzando la sua chiave privata f_B^{-1}.

> *La **sicurezza** del sistema si fonda sul fatto che dalla conoscenza di f è sostanzialmente **impossibile** risalire alla conoscenza di f^{-1}.*

Un crittosistema viene definito dalla scelta di una particolare funzione invertibile f. Il software GPG supporta i seguenti sistemi crittografici:

- RSA: elaborato da R. Rivest, A. Shamir e L. Adleman, fonda la sua sicurezza sulla difficoltà di fattorizzare numeri con molte cifre; il brevetto del sistema è scaduto nel 2000;
- ElGamal: elaborato nel 1984 da T. Elgamal fonda la sua sicurezza sulla difficoltà del calcolo del logaritmo discreto;
- DSA: una variante rafforzata del sistema di Elgamal adottata nel 1991 dal governo degli Stati Uniti.

4.1.1 Creare una coppia di chiavi

Il comando per generare una coppia di chiavi (una pubblica e una privata) con GPG è

```
gpg --gen-key
```

Il primo problema è quale algoritmo usare. La scelta predefinita e la più usata consiste nell'utilizzare l'algoritmo DSA/ElGamal, che non è brevettato. Più precisamente con questa scelta verranno generate una coppia di chiavi per firmare documenti con il metodo DSA e per cifrarli con il metodo di ElGamal. Entrambi i metodi, come ricordato in precedenza, affidano la loro sicurezza all'impossibilità presunta di calcolare in tempi brevi il logaritmo discreto di un numero intero. Dal momento che abbiamo studiato principalmente il metodo RSA genereremo una coppia di chiavi di questo tipo.

Il secondo problema riguarda la lunghezza della chiave. In questo caso la scelta dipende dalle preferenze dell'utente che dovrà realizzare un compromesso tra sicurezza e tempo di calcolo: se una chiave è lunga il rischio di decrittazione di un messaggio intercettato si riduce a scapito del tempo di calcolo impiegato per cifrare e decifrare i documenti. Nel nostro esempio utilizzeremo il valore suggerito *di default* pari a 2048 bit.

Il terzo problema riguarda la durata della chiave. Per la maggior parte delle persone è sufficiente utilizzare l'opzione predefinita che prevede la generazione di chiavi senza data di scadenza.

Successivamente, il sistema chiederà di inserire nomi, commenti e indirizzi e-mail, che verranno usati per la costruzione della chiave; questi potranno essere modificati in parte anche successivamente. Infine, occorre scegliere una password (di solito si usa il termine *passphrase*, frase segreta, visto che sono ammessi gli spazi), che andrà immessa ogni volta che si useranno funzionalità che richiedono il ricorso alla chiave privata. Una buona passphrase ha le seguenti caratteristiche:

- è lunga,
- contiene caratteri speciali (non alfanumerici),
- è una parola speciale (non un nome comune),
- è molto difficile da indovinare (quindi niente nomi, date di nascita, numeri di telefono o di carta di credito, nomi di bambini, eccetera).

Si può migliorare la sicurezza anche usando 'Le MAiusCOLe aLTernATe allE mInusCOLe'. La passphrase *non* deve essere dimenticata: se ciò succedesse, non sarebbe più possibile usare la propria chiave privata, ad esempio per leggere i messaggi ricevuti. È cosa saggia anche generare un certificato che contiene queste informazioni e conservarlo con cura in un luogo sicuro.

Dopo aver immesso tutti i dati, il sistema inizerà a generare le chiavi. Questa operazione richiederà un po' di tempo, durante il quale il sistema deve raccogliere molti dati casuali. Si può agevolare la generazione di dati casuali ad esempio lavorando su un'altra finestra con altri programmi. Ogni chiave generata è diversa dalle altre: se si genera una chiave e dopo cinque minuti se ne genera un'altra fornendo gli stessi dati (nome, email, passphrase, eccetera), si otterrà una chiave diversa.

Esempio concreto

Un esempio concreto di generazione di una coppia di chiavi con GPG è il seguente

```
[felice@localhost ~] gpg --expert --gen-key
gpg (GnuPG) 1.4.0;
Copyright (C) 2004 Free Software Foundation, Inc.
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to
redistribute it under certain conditions.
See the file COPYING for details.

gpg: portachiavi '/home/felice/.gnupg/secring.gpg' creato
gpg: portachiavi '/home/felice/.gnupg/pubring.gpg' creato
Per favore scegli che tipo di chiave vuoi:
 (1) DSA and Elgamal (default)
 (2) DSA (firma solo)
 (3) DSA (set your own capabilities)
```

```
 (5) RSA (firma solo)
 (7) RSA (set your own capabilities)
Cosa scegli? 7

Possible actions for a RSA key: Sign Encrypt Authenticate
Current allowed actions: Sign Encrypt

 (S) Toggle the sign capability
 (E) Toggle the encrypt capability
 (A) Toggle the authenticate capability
 (Q) Finished

Cosa scegli? Q
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048) 2048
La dimensione richiesta della chiave è 2048 bit
Per favore specifica per quanto tempo la chiave sarà valida.
       0 = la chiave non scadrà
    <n>  = la chiave scadrà dopo n giorni
    <n>w = la chiave scadrà dopo n settimane
    <n>m = la chiave scadrà dopo n mesi
    <n>y = la chiave scadrà dopo n anni
Chiave valida per? (0) 0
Key non ha scadenza
Is this correct? (y/N) y

You need a user ID to identify your key;
the software constructs the user ID
from the Real Name, Comment and Email Address in this form:
"Heinrich Heine (Der Dichter) <heinrichh@duesseldorf.de>"

Nome e Cognome: Andrea Centomo
Indirizzo di Email: andrea.centomo@istruzione.it
Commento: Scuola
Hai selezionato questo User Id:
"Andrea Centomo (Scuola) <andrea.centomo@istruzione.it>"

Modifica (N)ome, (C)ommento, (E)mail oppure (O)kay/(Q)uit? O
Ti serve una passphrase per proteggere la tua chiave segreta.

Dobbiamo generare un mucchio di byte casuali.
E' una buona idea eseguire qualche altra azione
(scrivere sulla tastiera, muovere il mouse, usare i
dischi) durante la generazione dei numeri primi;
questo dà al generatore di numeri casuali migliori
```

```
possibilità di raccogliere abbastanza entropia.
.+++++
+++++
gpg: /home/felice/.gnupg/trustdb.gpg: creato il trustdb
gpg: key 0697E14B marked as ultimately trusted
chiavi pubbliche e segrete create e firmate.

gpg: controllo il trustdb
gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model
gpg: depth: 0  valid:   1  signed:   0
   trust: 0-, 0q, 0n, 0m, 0f, 1u
pub   2048R/0697E14B 2006-03-09
Key fingerprint = 134C DB27 3761 7EA4 A651
                  4402 4C03 8FB2 0697 E14B
uid Andrea Centomo (Scuola) <andrea.centomo@istruzione.it>
```

Il procedimento ricalca quanto visto in precedenza. Ci limitiamo solo a osservare alcuni aspetti importanti:

1. alcuni file sono stati creati nella directory `/home/felice/.gnupg` (directory nascosta);
2. è stata creata l'impronta digitale *Key Fingerprint* della chiave pubblica;
3. l'utilizzatore è

   ```
   Andrea Centomo (Scuola) <andrea.centomo@istruzione.it>
   ```

4.1.2 Esportare e importare una chiave pubblica

Il comando per esportare la chiave di un utente è

```
gpg --export [UID]
```

Se non si indica un UID (identificativo dell'utente), verranno esportate tutte le chiavi presenti nel portachiavi. In molti casi è consigliabile usare l'opzione `-a` per scrivere la chiave in un file ASCII a 7 bit, invece che in un file binario. Per allargare il proprio orizzonte e permettere ad altri di inviare messaggi in modo sicuro, occorre esportare la propria chiave pubblica e pubblicarla sulla propria home page oppure caricarla su un *key server*[2] o ancora usando altri metodi.

[2] Un *key server* è un computer in rete adibito a ospitare le chiavi pubbliche di tutti gli utenti che lo desiderano.

Esempio concreto

Con riferimento all'esempio precedente vediamo come esportare la chiave pubblica dell'utente Andrea Centomo:

```
gpg -a --export Andrea Centomo > centomopubkey.txt
```

Mediante il comando > l'output non viene scritto a video ma ridiretto nel file `centomopubkey.txt`. La chiave pubblica è ora visualizzabile aprendo, con un qualsiasi editor di testo, il file `centomopubkey.txt`:

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.4.0 (GNU/Linux)

mQEMBEQQWmMBCADLaOBWyardPV9hZOBoGOsip8fN733vzCAtPZ/V5onHA
9z85kc9c7wqDH/o5l/5qpoTCKOYMaOCud0+DBLi4uT4LwJxwQ+vL/llvM
dreb9qm2AE4DvlotQr45dsUZbwUVCsCk1uzpUnpJ15c2p0LuU469Xdo59
IbiFMWR7hJZAp7xl8twjF8olrUoczYZfXn/oI9OGIiqVdcuePx20BaF1S
/F0e8D6STPBNxfFcwMjpXgRtRZSDyvENjfCUrKOyopue+gcKNsFq8AReV
Oi7Bu7Uq3riawpDG9PX7d8ciCnt8RhhMWgmOHSp7qX9oQlhH4AqTsnsx2
qbK49r4wQHVMm7AAkBAbQ2QW5kcmVhIENlbnRvbW8gKFNjdW9sYSkgPGF
uZHJlYS5jZW50b21vQGlzdHJ1emlvbmUuaXQ+iQE0BBMBAgAeBQJEEFpj
AhsPBgsJCAcDAgMVAgMDFgIBAh4BAheAAAoJEEwDj7IGl+FLOxUIAKs86
ruElHNq6rjAyc/BYyCEfKmY6ki4wqRPBWcm2FNGPM8HKMjwDHpExralhn
EEwsVbraJo3cLFxjpCm2cpH7svZxhRtpNkdv47hVu6HWxmS8EoZdx6rR+
FsrLg9HnFq2K0XrhCcImLjRFVlUvi5iw54Kv+DorCBG5Dj98+VUKpvO91
dZmcd1CbD7kt3cQl0IEkLC8MzB8LrmTVZu0JpPhJ2aJ2rSv4Y+t02cy9/
LrYSW9myxuo1HX0EVeHmuy22Plz4H9EvKXaH53gjkVc3HTOmPqaDEflls
VuUUMHh6krGhqcIzBDYvcd1RuOH3gjrzpMnG1lvzsuhAbbkxFq/Rw==nm
tp
-----END PGP PUBLIC KEY BLOCK-----
```

Quando si riceve la chiave pubblica di qualcuno, prima di usarla occorre importarla nel proprio portachiavi (si può farlo anche per più di una chiave alla volta). Il comando è il seguente:

```
gpg --import [nomefile]
```

dove si suppone che la chiave da importare sia stata scritta nel file che ha nome `nomefile`.

4.1.3 Amministrazione delle chiavi

Il sistema GPG comprende dei file che servono per immagazzinare tutte le informazioni che accompagnano le chiavi. Con il comando

```
gpg --list-keys
```

verranno mostrate tutte le chiavi esistenti. Per vedere le firme si digita

```
gpg --list-sigs
```

mentre per vedere le impronte digitali si digita

```
gpg --fingerprint
```

Vedere le impronte digitali serve ad assicurarsi che la chiave appartenga davvero alla persona che sostiene di esserne il proprietario (ad esempio al telefono). L'output di questo comando è una breve lista di numeri. Per vedere la lista delle chiavi private si usa:

```
gpg --list-secret-keys
```

Si noti che non ha alcuna utilità vedere firme o impronte digitali di chiavi private!
Per cancellare una chiave pubblica si usa:

```
gpg --delete-key UID
```

Per cancellare una chiave privata si usa:

```
gpg --delete-secret-key
```

C'è un altro comando importante per la gestione delle chiavi:

```
gpg --edit-key UID
```

Usando questo comando è possibile modificare (tra le altre cose) la data di scadenza di una chiave, aggiungere UID o firmare una chiave (ovviamente per questo è necessaria la propria passphrase). Dopo aver eseguito il comando `-edit-key` si otterrà un prompt interattivo da cui digitare i comandi successivi.

4.1.4 Firma di documenti

Supponiamo di avere un documento e di volerlo firmare con la propria chiave. Ci sono due modi:

```
gpg -s (o --sign) [documento]
```

Durante questa operazione i dati vengono anche compressi, quindi il risultato non sarà leggibile. Per avere un risultato leggibile si può usare:

```
gpg --clearsign [documento]
```

In questo modo si firmeranno i dati lasciandoli leggibili. Quando dei dati sono stati firmati, la firma viene verificata. È possibile verificare le firme con il comando

```
gpg [--verify] [dati]
```

ma ovviamente per fare questo è necessario possedere la chiave pubblica del mittente.

Esempio concreto

Supponiamo di aver scritto nel file `voti.txt` il seguente testo

```
Risultati Compito 12 Febbraio 2006

Bianchi Luigi 23
Rossi Mario   18
Verdi Paolo   12

Visione elaborati: 23 Febbraio 2006 ore 15 Aula B.
```

e di volerlo inviare a una lista di studenti. Possiamo firmarlo

```
gpg --clearsign voti.txt
```

ottenendo il file firmato `voti.txt.asc` il cui contenuto, visualizzabile con un editor di testo, è

```
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1

Risultati Compito 12 Febbraio 2006

Bianchi Luigi 23
Rossi Mario   18
Verdi Paolo   12

Visione elaborati: 23 Febbraio 2006 ore 15 Aula B.
-----BEGIN PGP SIGNATURE-----
Version: GnuPG v1.4.0 (GNU/Linux)

iQEVAwUBRBBbXEwDj7IGl+FLAQKbjQf+ME3W1OCPz127DfyQ3fWVIb8
Ldac/Mk2p90JOSZQDoZl29Nt56m5a9qxdm2IHWDzN94GxUH1oq7ip+l
O4akakRiiPGvMdWiqsJtanH3Ac5MoblROtzPER/uEKAq0k0vAJzzL0L
t+bhVeZ1eJlDSk8JdlV0s5TtAX5f63oN8vhERtzsNFWAhjh0v3Dyjki
Uk4C2ZjN3GEf0Ralwf1TVlH8gYUq6gJjEUqHss46c3rokAnQclCfNWJ
HgRXm4iddPeOEKIpf12SvVIV897GI+6/D7LO5DeJcVzwK3DjrOY4yGA
oaQdGIUOuA8M5UMT5mGP6SaTbYt7X/db7ggFzqAmCpmQ===CWbi
-----END PGP SIGNATURE-----
```

Osserviamo che per produrre la firma è stata usata la funzione *hash* nota con il nome 'SHA1'. Lo studente che possiede la chiave pubblica del professore può verificare l'autenticità del documento con il comando visto in precedenza.

```
gpg --verify voti.txt.asc
gpg: Signature made gio 09 mar 2006 17:44:12 CET using RSA
   key ID 0697E14B
gpg: Good signature from "Andrea Centomo
   (Scuola) <andrea.centomo@istruzione.it>"
```

4.1.5 Firma delle chiavi

Il principale tallone d'Achille del sistema è l'autenticità delle chiavi pubbliche: se si usa una chiave pubblica contraffatta si può dire addio alla crittografia sicura. Per evitare questo rischio, c'è la possibilità di firmare le chiavi, ossia di porre la propria firma digitale sulla chiave, certificandone la validità. In pratica, la firma certifica che lo user ID menzionato nella chiave corrisponde alla persona che possiede la chiave. Una volta che si è certi di questo, si può usare la chiave in tutta sicurezza. Per firmare una chiave, si usa il comando

```
gpg --edit-key UID
```

e successivamente il comando `sign`.

Importante. Bisogna firmare una chiave solo quando si è *assolutamente certi* della sua autenticità! Questa condizione può verificarsi quando si è ricevuta la chiave direttamente da una persona (ad esempio durante un *key signing party*, un raduno per la firma delle chiavi) o quando la si è ricevuta per altri mezzi e se ne è controllata l'autenticità col metodo dell'impronta digitale (ad esempio al telefono). Non si dovrebbe mai firmare una chiave a priori.

GPG calcola la validità delle chiavi basandosi sulle firme disponibili e sui valori di fiducia nel proprietario (*ownertrust*). La fiducia nel proprietario di una chiave rappresenta la fiducia che si ripone nella capacità del proprietario di firmare correttamente altre chiavi. I valori possibili sono:

- 1 = Indefinito
- 2 = Nessuna fiducia
- 3 = Fiducia marginale
- 4 = Fiducia completa

Così, se non si ha fiducia nel proprietario di una chiave, le eventuali firme apposte da quest'ultimo su un'altra chiave verranno ignorate durante il calcolo del valore di validità della chiave. Le informazioni sulla fiducia non sono immagazzinate negli stessi file che contengono le chiavi, ma in un file separato.

4.1.6 Cifratura e decifratura di messaggi

Supponiamo che un utente che possiede la chiave pubblica di Andrea Centomo desideri spedirgli il messaggio cifrandolo con il sistema RSA. Supponiamo per comodità che il messaggio sia scritto nel file `messaggio.txt`. L'utente può utilizzare il comando:

```
gpg --output cifrato.gpg --encrypt
--recipient andrea.centomo@istruzione.it
messaggio.txt
```

Osserviamo che il messaggio cifrato verrà scritto nel file `cifrato.gpg` e che il destinatario viene precisato attraverso l'indirizzo email associato alla chiave pubblica. Per decifrare il messaggio Andrea Centomo utilizzerà il comando:

```
gpg --output decifrato.txt --decrypt cifrato.gpg
```

Capitolo 5

Complementi

In questo capitolo sono contenute le rielaborazioni dei contributi di alcuni studenti che hanno seguito in modo particolarmente serio le lezioni del Progetto Lauree Scientifiche.

5.1 Poligoni stellati e Teorema di Wilson

In questo paragrafo si espone un'elegante dimostrazione del Teorema di Wilson che si fonda su alcune considerazioni geometriche relative ai poligoni stellati regolari.

5.1.1 Poligoni stellati regolari

Su una circonferenza qualsiasi disegnamo cinque punti equispaziati e immaginiamo di eseguire la seguente costruzione: tramite un segmento congiungiamo uno di questi punti con il punto che si trova due posti oltre, ruotando sulla circonferenza in senso orario; congiungiamo quindi il punto di arrivo al punto che si trova due posti oltre ruotando sulla circonferenza sempre in senso orario e via di seguito fino a ritornare al punto iniziale. Quello che si ottiene è il pentagramma rappresentato in Figura 5.1.

La costruzione sopra descritta ci ha permesso di disegnare, senza mai staccare la matita dal foglio, un poligono regolare stellato con cinque vertici. Indichiamo come d'uso questo poligono con il simbolo di Schläfli $\{5,2\}$ dove 5 indica il numero di vertici e 2 lo spostamento utilizzato per congiungere i punti.

Se proviamo a ripetere l'intero procedimento partendo da sei punti, sempre con spostamento pari a due, il poligono che si ottiene non ha sei vertici e, soprattutto, non è univocamente determinato. Infatti congiungendo successivamente i diversi

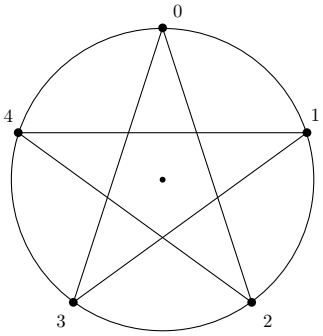

Figura 5.1: Pentagramma

punti otterremo due triangoli equilateri diversi, a seconda che si inizi da un punto di indice pari o dispari.

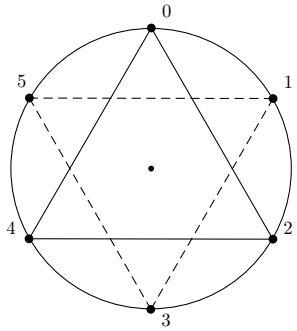

Figura 5.2: Triangoli equilateri

Il problema generale che ora ci poniamo è il seguente: dati n punti equispaziati su una circonferenza e uno spostamento k, con $n > k > 2$, quali condizioni devono soddisfare i numeri n e k per essere sicuri *a priori* di ottenere un poligono stellato di esattamente n vertici? La risposta è semplice e si basa sul seguente

Lemma 48. *Sia $k \in \mathbb{Z}_n$ un intero coprimo con n, allora l'insieme di numeri interi $A = \{[k]_n, [2k]_n, \ldots, [nk]_n\}$ coincide con $\mathbb{Z}_n$.*

Dimostrazione. La dimostrazione è identica a quella del Lemma 36. □

Se ora consideriamo su una circonferenza i punti equispaziati P_0, P_1, …, P_n, e se partendo da P_0 iniziamo a realizzare la costruzione descritta all'inizio del

paragrafo per disegnare un poligono stellato, dovrebbe essere chiaro dal Lemma precedente che otterremo un poligono regolare stellato $\{n,k\}$ solo se n e k sono coprimi. Possiamo allora, seguendo [3] dare la seguente definizione.

Definizione 49. Siano n e k, con $n > k > 2$, due interi positivi coprimi. Su una circonferenza consideriamo n punti equispaziati e fissato uno di essi P_0 lo congiungiamo con un segmento al punto P_1 che si trova k posti oltre, muovendosi in senso orario. Congiungiamo quindi P_1 con il punto P_2 che si trova k posti oltre muovendosi in senso orario e via di seguito fino a ritornare al punto iniziale P_0. Il poligono che si ottiene con questa procedura iterativa si dice poligono regolare stellato e si indica con la notazione $\{n,k\}$.

Il simbolo di Schläfli $\{n,k\}$ contiene una coppia di numeri: n indica il numero di vertici del poligono e k indica che ogni vertice del poligono è congiunto con il vertice che si trova k posizioni oltre, ruotando in senso orario.

Fissato un intero positivo n il valore assunto dalla funzione di Eulero $\varphi(n)$ definisce il numero di poligoni stellati possibili $\{n,k\}$. Osserviamo tuttavia che alcuni di essi hanno lo stesso aspetto geometrico e che il numero di poligoni stellati *diversi* è dato da $\varphi(n)/2$.

5.1.2 Poligoni stellati e aritmetica

I poligoni stellati sono legati in diversi modi all'aritmetica modulare e, più in generale, alla Teoria dei Numeri. In questo paragrafo, seguendo H. S. M. Coxeter [3], proponiamo una dimostrazione geometrica del Teorema di Wilson[1] e una dimostrazione geometrica del Piccolo Teorema di Fermat.

Teorema 50 (Wilson). *Sia p un numero intero, allora p è primo se e solo se*

$$(p-1)! \equiv -1 \pmod{p}$$

o equivalentemente se $p \mid ((p-1)!+1)$.

Dimostrazione. Supponiamo che p sia primo. Fissati p punti equispaziati su una circonferenza di centro C consideriamo l'insieme $\mathscr{P}$ formato da tutti i poligoni con p vertici che si ottengono congiungendo uno dopo l'altro i p punti dati con un segmento. L'insieme $\mathscr{P}$ contiene $(p-1)!$ poligoni. Di questi $\varphi(p) = p-1$ appartengono al sottoinsieme $\mathscr{S} \subseteq \mathscr{P}$ dei poligoni stellati regolari e sono invarianti per rotazioni intorno a C di un qualsiasi angolo multiplo di $\alpha = 2\pi/p$. Un poligono che non sia regolare stellato non è invece invariante per l'azione delle stesse rotazioni. Quindi, dato uno qualsiasi di essi, il poligono che si ottiene ruotandolo di un angolo multiplo

[1] Il teorema proposto inizialmente da J. Wilson e pubblicato da E. Waring nel 1770, venne dimostrato da J. L. Lagrange nel 1773.

di α non coincide con il poligono di partenza. In altri termini possiamo pensare che l'insieme $\mathscr{P}\setminus\mathscr{S}$ sia formato da un certo numero n di poligoni non regolari stellati a cui si devono aggiungere, per ciascuno di essi, i $p-1$ poligoni ottenuti per rotazione intorno a C di un angolo multiplo di α. Il numero totale dei poligoni che hanno p vertici sarà allora dato da:

$$np + (p-1) = (n+1)p - 1 = (p-1)!$$

Dall'ultima uguaglianza si ha direttamente la tesi $p \mid [(p-1)!+1]$. □

Osserviamo che il Teorema di Wilson permette, almeno in linea di principio, di verificare la primalità di un numero intero *senza* dover ricorrere alla sua scomposizione in fattori primi. Ad esempio, si verifica subito che 7 è primo calcolando $6!+1 = 721$ e notando che $7 \mid 721$. Tuttavia esso risulta nel complesso inutile come criterio pratico di verifica della primalità di un numero in quanto il calcolo del fattoriale di numeri interi con molte cifre è computazionalmente proibitivo.

Teorema 51 (Piccolo Teorema di Fermat). *Sia $p > 1$ un numero primo, allora per ogni numero $a \in \mathbb{Z}_p^*$ si ha*

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

o equivalentemente $p \mid (a^{p-1}-1)$.

Dimostrazione. Consideriamo i poligoni stellati $\{p,1\}$ e $\{p,a\}$ e osserviamo che hanno gli *stessi* vertici a meno dell'ordine con cui essi vengono congiunti. Il primo poligono viene costruito congiungendo nell'ordine i vertici $\{0,1,2,\ldots,p-1,0\}$ il secondo i vertici $\{0,a,2a,\ldots,(p-1)a,0\}$. Quindi avremo

$$a \cdot 2a \cdot \ldots \cdot (p-1)a \equiv 1 \cdot 2 \cdot \ldots \cdot (p-1) \pmod{p}$$

ossia $a^{p-1}(p-1)! \equiv (p-1)! \pmod{p}$. Quindi $p \mid (a^{p-1}-1)(p-1)!$ e dal momento che p non divide $(p-1)!$ deve essere che $p \mid (a^{p-1}-1)$. □

5.1.3 Poligoni stellati impropri

Possiamo estendere il concetto di poligono regolare stellato ai casi in cui n e k non siano tra loro coprimi? Una costruzione geometrica generale che include la situazione in cui viene a mancare la coprimalità tra n e k è la seguente:

(a) dati n punti equispaziati su una circonferenza si considera un vertice qualsiasi e si congiunge a quello che viene k posti dopo nel senso orario;

(b) si procede come in (a) fino a che non si torna al vertice di partenza;

(c) se n e k sono coprimi la costruzione è conclusa, altrimenti quanto svolto fino al punto (b) non permette di connettere tutti gli n punti; si considera allora uno qualsiasi dei punti non connessi e si ripete la costruzione descritta al punto (a).

I poligoni che non sono regolari e stellati nel senso della Definizione 49 si dicono poligoni stellati regolari *impropri*. Diversi esempi di questo genere di poligoni si ritrovano nello studio dell'Arte.

Esempio 52. La stella di Lakshmi $\{8,2\}$ è un poligono regolare stellato improprio.

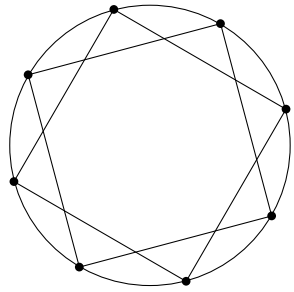

Figura 5.3: Stella di Lakshmi

5.2 Il codice di Vigenère in JavaScript

JavaScript è un linguaggio di scripting orientato agli oggetti comunemente usato nei siti web. Fu originariamente sviluppato da B. Eich della Netscape Communications con il nome di Mocha e successivamente di LiveScript, ma in seguito è stato rinominato JavaScript ed è stato formalizzato con una sintassi più vicina a quella del linguaggio Java di Sun Microsystems.

Attraverso il linguaggio JavaScript si possono implementare, tra le infinite cose, anche alcuni codici classici della crittografia potendo disporre immediatamente di una interfaccia web per l'inserimento interattivo dei diversi dati necessari per il loro funzionamento. In quanto segue viene riportato uno script commentato che implementa il codice di Vigenère. Evitiamo al lettore la noiosa descrizione tecnica dello script invitandolo invece a copiare e incollare il suo contenuto in un file, con nome ad esempio `vigenere.html`, e quindi a utilizzarlo con un qualsiasi navigatore Internet.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

<HTML>
        <TITLE>Vigenère</TITLE>

<SCRIPT language=JavaScript>
function Vigenere(input, chiave, output, crypter)
{
 var alfabeto = document.form.alfabeto.value,
 alfabetoLunghezza = alfabeto.length;
 input.value = input.value.toUpperCase();

  // Verifica della chiave:
 if (chiave.value == null) {chiave.value = "";}
 //chiave.value = chiave.value.toUpperCase();
 var chiaveLunghezza = chiave.value.length;
 var chiaveBianca = "";

 for (var i = 0; i<chiaveLunghezza; i++) {
 var chiave_char = alfabeto.indexOf(chiave.value.charAt(i));
 if (chiave_char>-1)
 {chiaveBianca +=alfabeto.charAt(chiave_char)};
 }

 chiave.value = chiaveBianca;
 chiaveLunghezza = chiave.value.length;
 if (chiaveLunghezza == 0) {
        alert ("Inserire la vostra chiave.");
        chiave.value = "Inserire la chiave qui!";
        chiaveLunghezza = 1; }

// (De)crittaggio
output.value = "";
var chiave_index = 0; // parti dall'inizio della chiave

for (i=0; i<input.value.length; i ++) {

//nella var "Lettera", va il carattere alla posizione i
var Lettera = input.value.charAt(i);

//nella var "nLettera" va il numero della lettera dell'alfabeto
//alla posizione i(parte da 0)
var nLettera = alfabeto.indexOf(Lettera);
```

```
if (nLettera>−1){

//se il codice è dentro l'alfabeto
//e bisogna crittare, aggiungo al numero della lettera
// il numero della lettera nella posizione della chiave

if (crypter) nLettera +=
    alfabeto.indexOf(chiave.value.charAt(chiave_index));
//se bisogna decrittare: sottraggo al numero della lettera
// il numero della lettera nella posizione della chiave

else nLettera −=
    alfabeto.indexOf(chiave.value.charAt(chiave_index));
//aggiungo al numero della lettera in output il numero totale
//di segni dell'alfabeto
nLettera += alfabetoLunghezza;

//torno indietro ogni numero totale di segni dell'alfabeto
nLettera %= alfabetoLunghezza;

//aggiungo alla stringa "output" la lettera codificata

output.value+=alfabeto.charAt(nLettera);

//al prossimo giro dico di avanzare di una
// lettera nella chiave
chiave_index = (chiave_index+1) % chiaveLunghezza; }
 }
}
</SCRIPT>

<BODY>
<FORM name=form>
<P><STRONG>Metodo di Vigenère</STRONG><BR>
Inserire la chiave, il messaggio da crittare o decrittare
nell'apposito campo. Il programma accetta solo i caratteri
inseriti nel campo "alfabeto".</P>

<P>
Alfabeto:<BR>
<TEXTAREA name="alfabeto" wrap=virtual cols=70>
ABCDEFGHIJKLMNOPQRSTUVWXYZ</TEXTAREA>
</P>
<!−−0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ:;@#*^'−−>
```

```
<P>
Testo da codificare:<BR>
<TEXTAREA name=input rows=6 wrap=virtual cols=70></TEXTAREA>
</P>
Testo codificato:<BR>
<TEXTAREA name=output rows=6 wrap=virtual cols=70></TEXTAREA>
</P>
<P>
Chiave:
<TEXTAREA name=chiave wrap=virtual cols=70></TEXTAREA>
</P>
<P>
<INPUT onclick="Vigenere(input,chiave,output,true)" type=button
    value="Cifra con Vigenère" name=button>
<INPUT onclick="Vigenere(output,chiave,input,false)" type=button
    value="Decifra da Vigenère" name=button2>
<INPUT onclick="input.value=''; output.value=''; chiave.value=''"
   type=button value="Cancella tutto" name=button3>
</P>

<P>Copyright 2007 Alessandro Dal Maso.</P>
</FORM>
</BODY>
</HTML>
```

5.3 Il Teorema dei Numeri Primi

Un argomento che ha sempre affascinato i matematici e di chiaro interesse per la crittografia riguarda la distribuzione dei numeri primi. Apparentemente i numeri primi si presentano distribuiti nell'insieme degli interi in modo caotico. Analisi più approfondite dimostrano invece l'esistenza di alcune forme di regolarità. Riscoprendo un risultato adombrato da Legendre nei suoi lavori sulla Teoria dei Numeri, il giovane K. F. Gauss scrisse sulla copia di un manualetto di logaritmi da lui usato quando era solo quattordicenne la seguente annotazione [2]:

$$\text{Primzahlen unter } a \quad (a = \infty) \quad \frac{a}{la}$$

che possiamo tradurre come

$$\pi(a) \sim \frac{a}{\log a} \qquad \text{per } a \to +\infty$$

dove $\pi(a)$ indica il numero dei primi fino ad a. Si tratta dell'enunciato del famoso Teorema dei Numeri Primi che possiamo riformulare con linguaggio più preciso da un punto di vista numerico [7] come segue.

Teorema 53 (dei Numeri Primi). *Indicata con*

$$\mathrm{li}(a) = \int_2^a \frac{1}{\log t}\, dt$$

la funzione logaritmo integrale di a, si ha, per $a \to +\infty$,

$$\pi(a) = \mathrm{li}(a) + O\left(ae^{-\sqrt{\log a}}\right)$$

dove $\pi(a)$ *indica il numero di primi minori di a.*

Nel 1896 i francesi J. Hadamard e C. J. de la Vallée Poussin, indipendentemente, riuscirono a dimostrare il Teorema dei Numeri Primi. La dimostrazione, che esula dagli scopi di questa introduzione, usa l'idea fondamentale introdotta da Riemann nel 1858 di studiare la distribuzione dei primi mediante l'analisi complessa. Notiamo solamente che, da un punto di vista euristico, il Teorema dei Numeri Primi consente di aspettarci che, per a abbastanza grande, esista un numero primo ogni $\log a$ interi circa, ossia, a conforto di chi usa tecniche crittografiche in cui le chiavi sono rappresentate da numeri primi, che i numeri primi abbondano.

Nella Tabella 5.1 vengono confrontati a titolo di esempio il numero $\pi(a)$ di primi fino a un certo numero a, il numero di primi secondo la congettura di Gauss e il corrispondente valore della funzione logaritmo integrale.

a	$\pi(a)$	$a/\log a$	$\mathrm{li}(a)$
10^3	168	145	178
10^4	1229	1086	1246
10^5	9592	8686	9630
10^6	78498	72382	78628
10^7	664579	620420	664918
10^8	5761455	5428681	5762209

Tabella 5.1: Densità dei primi

Bibliografia

[1] G. Alberti, Aritmetica finita e crittografia a chiave pubblica. *Raccolta "Ricordando Franco Conti", Classe di Scienze, Scuola Normale Superiore di Pisa*, 2004.

[2] C. Boyer, *Storia della Matematica.* Mondadori, 2000.

[3] H. S. M. Coxeter, *Introduction to Geometry.* Wiley, 1965.

[4] M. Fischer v. Mollard, *Gnu Privacy Guard Mini Howto*, 2003. Disponibile in italiano al sito web `http://www.gnupg.org/(it)/documentation/howtos.html`.

[5] S. Lang, *Algebra.* Springer Verlag, 2005.

[6] A. Languasco, A. Zaccagnini, Alcune proprietà dei numeri primi. 2005. Sito web Bocconi-Pristem `http://matematica.unibocconi.it/LangZac/home.htm`.

[7] A. Languasco, A. Zaccagnini, *Introduzione alla Crittografia.* Hoepli, 2005.

[8] A. Languasco, A. Zaccagnini, *Crittografia.* Cleup, 2006.

[9] S. Singh, *Codici & Segreti.* BUR, 2005.

[10] A. Zaccagnini, L'importanza di essere primo. *Raccolta "Ricordando Franco Conti", Classe di Scienze, Scuola Normale Superiore di Pisa*, 2004.

[11] A. Zaccagnini, Cryptographia ad usum delphini. 2005. Sito web dell'autore `http://www.math.unipr.it/~zaccagni/psfiles/papers/CryptoDelph.pdf`.