Fattorizzazione e criptosistemi a chiave pubblica

DIDATTICA DELLE SCIENZE, 137, ottobre 1988.

Riassunto. Dopo alcuni richiami sulla scomposizione in fattori primi di numeri interi ed il calcolo "modulo n", l'Autore affronta il tema della criptografia, in relazione alle sue applicazioni nelle comunicazioni di messaggi per via elettronica.

Introduzione.

Un numero primo è un intero, maggiore a uno, che è divisibile soltanto per uno e per se stesso. Eccone alcuni: 2, 3, 5, 7, 11, 13, ... Ogni intero positivo si può scomporre, in modo unico (a meno dell'ordine dei fattori), nel prodotto di numeri primi. Per esempio:

$$60 = 2^2 \cdot 3 \cdot 5;$$
$$10001 = 73 \cdot 137.$$

Oggigiorno, con l'avvento dei calcolatori elettronici, si cercano metodi efficienti per trovare la scomposizione in fattori primi di un dato intero n. Come vedremo, questo problema è di interesse sia pratico che teorico. Esso si spezza, in modo naturale, in due parti:

1) decidere se un dato numero è primo o no. Questo è il cosiddetto problema dei *test di primalità.*

2) determinare esplicitamente una qualche scomposizione in fattori (non banali) di un numero composto. Si presenta cioè il problema della *fattorizzazione.*

Anche se chiaramente legati l'uno all'altro, questi due problemi sono di natura differente. Allo stato presente della conoscenza de delle tecniche, il primo risulta relativamente facile, mentre il secondo sembra essere piuttosto difficile. Citiamo un passo dal discorso di H. W. Lenstra jr. al Congresso Internazionale di Matematica, tenutosi a Berkeley (USA) nel 1986:

> *"Supponiamo di aver dimostrato che due numeri p e q, di circa* 100 *cifre, sono primi; con gli odierni test di primalità la cosa è molto semplice. Supponiamo inoltre che per sbaglio, i numeri p e q vadano perduti nella spazzatura e che ci rimanga invece il loro prodotto $p \cdot q$. Come fare per risalire a p e q? Deve essere sentito come una sconfitta della matematica il fatto che la cosa più 'promettente' che possiamo fare sia di andare a cercare nell'immondizia o di provare con tecniche mnemo-ipnotiche ..."*

Per dare un'idea di ciò che oggigiorno è possibile su un grosso calcolatore, diciamo che si possono fattorizzare numeri che raggiungono le 90 cifre decimali. Naturalmente, fra essi ci sono anche numeri facili da fattorizzare; ad ogni modo, si tratta di casi piuttosto rari. In

generale, la fattorizzazione di un numero con più di 60 cifre decimali è difficile ed anche su di un grosso calcolatore essa richiede diverse ore. Il record mondiale è costituito da un intero di 93 cifre, fattorizzato da Te Riele, Lioen e Winter nell'aprile 1988. Ci sono volute 95 ore di calcoli su un super computer per trovare la scomposizione del numero seguente

$$2559041943566176656966919546515569274566618437762737512140975691256745820980515338664276477 7=$$

$$1284827442574221936870974393373403\cdot 199173979226268423344498336774040966135376386843488561780 59$$

C'è anche da notare che, sullo stesso computer e con lo stesso algoritmo, fattorizzare un intero col doppio delle cifre, richiederebbe all'incirca 200 000 anni. . .!

Il fatto che, per ora, quello dei test di primalitá sia un problema più fattibile si riflette, ad esempio, nel fatto che si riescono a trattare numeri molto più grandi: su un super computer, ci vogliono solo pochi minuti per decidere se un dato numero di 300 cifre è primo o no. Nel caso in cui il numero in questione non sia primo, in genere ciò non è rilevato dal test di primalità esibendone una fattorizzazione, ma invece facendo vedere che esso non soddisfa certe proprietà soddisfatte dai numeri primi. Vedremo piú avanti un esempio di tali criteri.

Occasionalmente è possibile determinare la primalità di numeri assai più grandi; ciò può verificarsi quando il numero in questione è di un tipo molto particolare. Ad esempio, fin dai tempi di Fermat (1601–1665) si è tantato di decidere la primalità dei numeri del tipo $2^n - 1$. È abbastanza facile far vedere che, affinchè $2^n - 1$ sia primo, è necessario che n stesso sia primo. Ciò, d'altra parte, non è sufficiente, come si vede esaminando già i primi esempi di numeri di questo tipo:

$$\begin{aligned} 2^2 - 1 &= 3, \\ 2^3 - 1 &= 7, \\ 2^5 - 1 &= 31, \\ 2^7 - 1 &= 127, \\ 2^{11} - 1 &= 2047, \\ 2^{13} - 1 &= 8191, \end{aligned}$$

Si ha infatti che $2^{11} - 1 = 2047 = 23 \cdot 89$. Padre Mersenne (1588–1648) fece una lista di primi p, per i quali $M_p = 2^p - 1$ risulta primo, da cui il nome di *primi di Mersenne*. Data la loro forma particolare, è relativamente facile decidere la primalità dei numeri M_p. Eulero dimostrò nel 1772 che $2^{31} - 1 = 2147483647$ è primo. Per più di un secolo esso rimase il più grosso numero primo conosciuto, fino a che, nel 1876, E. Lucas non dimostrò (a mano!) che è primo

$$2^{127} - 1 = 170141183460469231731687303715884105727.$$

Anche oggi il più grosso primo conosciuto è un numero di Mersenne: si tratta del numero a 65050 cifre dato da $2^{216091} - 1$ e scoperto da D. Slowinsky nel 1987.

Il fatto che, ora come ora, decidere la primalità di un numero sia assai più facile che fattorizzare un intero dello stesso ordine di grandezza, ha un'applicazione pratica molto

interessante. Nel 1976, R.L. Rivest, A. Shamir and L.M. Adleman hanno inventato un *criptosistema a chiave pubblica* detto *RSA*, che si basa su questo fatto: il loro sistema può essere usato per trasmettere messaggi segreti, pur restando, nel senso che verrà chiarito in seguito, pubblico. In tal modo, può essere usato dai vari utenti del sistema, come governo, banche, ... ecc., per mandarsi messaggi.

Le 'chiavi segrete' che vengono usate in questo sistema sono (oggi, nel 1988) degli interi di 200 cifre, N, che sono prodotto di due numeri primi (segreti) di circa 100 cifre; apparentemente, entrare nel sistema equivale ad essere in grado di fattorizzare N. Come indicato nella citazione di Lenstra, cone le tecniche odierne, ciò è virtualmente impossibile. Dunque, la sicurezza del sistema si basa sul fatto che, fino ad ora, nessuno ha ancora inventato un algoritmo efficiente per la fattorizzazione dei numeri. Un tale algoritmo potrebbe comunque esistere e chiunque trovasse un metodo veloce per fattorizzare i numeri interi, otterrebbe dalla sua scoperta, oltre alla soddisfazione scientifica, anche una notevole gratificazione economica.

In questo articolo, spiegheremo come funziona il *criptosistema a chiave pubblica RSA*. Daremo anche tutte le notazioni necessarie affinché il lettore possa costruire il suo *criptosistema* personale[1]: a tale scopo sarà spiegato un metodo efficiente per costruire numeri primi grandi a piacere. Per essere più precisi, diciamo che, per i numeri trovati, non si dimostrerà rigorosamente che essi sono primi, ma che hanno una grossa probabilità di esserlo, e ciò sarà sufficiente per i nostri scopi. Successivamente, discuteremo il ciptosistema e le sue 'caratteristiche pubbliche'[2].

Il calcolo 'modulo n".

In questo paragrafo spiegheremo come fare i calcoli *modulo* n, dove n è un intero positivo. La teoria che interviene fa parte dell'*algebra astratta*; al giorno d'oggi è ormai standard e viene insegnato al primo anno di un corso di laurea in matematica o di informatica. Nel nostro caso, sarà enfatizzato il punto di vista computazionale.

Nel calcolo *modulo* n, si fanno i calcoli sugli interi orinari (indicati con $\mathbf{Z}$), ma poi si identificano fra loro gli interi che differiscono per multipli di n. Si può verificare che le operazioni di addizione e di moltiplicazione sono compatibili con questa identificazione. Diremo più avanti qualche parola sulla divisione. Con questa identificazione, vengono ad esserci solo un numero finito di numeri, precisamente n. Vediamo, come esempio, il caso $n = 12$:

$$
\begin{array}{ccccccccc}
 & & -12 & = & 0 & = & 12 & = & 24 \\
 & & -11 & = & 1 & = & 13 & = & 25 \\
 & & -10 & = & 2 & = & 14 & = & \dots \\
 & & -9 & = & 3 & = & 15 & = & \dots \\
 & & -8 & = & 4 & = & 16 & = & \dots \\
 & & -7 & = & 5 & = & 17 & = & \dots \\
 & & -6 & = & 6 & = & 18 & = & \dots \\
 & & -5 & = & 7 & = & 19 & = & \dots \\
 & & -4 & = & 8 & = & 20 & = & \dots \\
 & & -3 & = & 9 & = & 21 & = & \dots \\
-14 & = & -2 & = & 10 & = & 22 & = & \dots \\
-13 & = & -1 & = & 11 & = & 23 & = & \dots
\end{array}
$$

Si vede che -10, 2 e 14 risulterano tutti identificati e ciò verrà indicato scrivendo $2 \equiv 14 \pmod{12}$ ('2 è congruo a 14 modulo 12). Similmente, si ha che $-2 \equiv 21 \pmod{12}$ ('-3 è congruo a 21 modulo 12), ecc.

L'insieme dei numeri dopo l'identificazione si denote con:

$$\mathbf{Z}/n\mathbf{Z},$$

ed è chiamato l'*anello degli interi modulo n*. Gli elementi di questo anello sono detti le *classi resto modulo n*. L'esempio visto sopra, per $n = 12$, è ben noto nella vita quotidiana: esso corrisponde alla 'aritmetica dell'orologio". Tutti sanno che 3 ore dopo le 11 sono le 2 e ciò equivale a dire che:

$$11 + 3 \equiv 2 \pmod{12}.$$

Molti test di primalità e algoritmi di fattorizzazione non sono altro che particolari calcoli su interi *modulo n*, dove n è il numero in esame. Tali calcoli si possono eseguire in modo efficiente anche su un computer, dove il calcolo 'modulo n' risulta altrettanto semplice come il calcolo sugli interi ordinari. Facendo calcoli modulo n sul computer, in generale si sceglie un insieme finito R di rappresentanti per le classi resto. Una scelta conveniente è per esempio:

$$R = \{0, 1, 2, \ldots, n-2, n-1\},$$

e si vede che ogni intero è congruo esattamente ad uno degli elementi di R. Infatti, se $N \in \mathbf{Z}$ è un intero arbitrario, si a q la parte intera del quoziente $\frac{N}{n}$, ossia l'intero più grosso minore o uguale a $\frac{N}{n}$. Sia $r = N - qn$. Abbiamo ora:

$$r \equiv N \pmod{n} \qquad \text{e} \qquad 0 \leq r < N,$$

o, in altre parole, abbiamo che $r \in R$. Il processo che associa ad un numero N il corrispondente r, si chiama *riduzione modulo n*. Può essere eseguito agevolmente al calcolatore: si valuta il quoziente $\frac{N}{n}$, come numero reale, e poi se ne prende la parte intera.

Nella maggior parte dei linguaggi di programmaziuone c'è una funzione chiamata di solit `int` oppure `trunc`, che calcola la parte intera di un numero (purtroppo, in molto implementazioni, la scelta del rappresentante di N modulo n viene fatta dipendere anche dal degno di N); in alcuni linguaggi, come ad esempio PASCAL, c'è addirittura la funzione $m \pmod{n}$, che restituisce la classe resto modulo n di m in R.

Per sommare due elementi di R, si esegue semplicemente la somma come per gli interi ordinari e poi si prende la rappresentazione in R di questa somma. Lo stesso si fa per il prodotto. Ecco alcuni esempi, sempre per $n = 12$:

$$7 + 8 = 15 \equiv 3 \pmod{12},$$
$$7 \cdot 8 = 56 \equiv 8 \pmod{12}.$$

Il vantaggio del calcolo modulo n sta nel fatto che, scegliendo rappresentanti piccoli in ogni fase dei calcoli, si può sempre evitare di lavorare con numeri grossi. Per esempio, dato che si ha

$$7^2 = 49 \equiv 1 \pmod{12},$$

risulta

$$7^{101} = (7^2)^{50} \cdot 7 \equiv 1^{50} \cdot 7 \equiv 7 \pmod{12}.$$

Discutiamo ora in maggior dettaglio l'algoritmo per il calcolo delle potenze modulo n. È possibile calcolare $a^m \pmod n$ per m intero positivo, per mezzo di successive elevazioni al quadrato e moltiplicazioni, usando la rappresentazione binaria di m; questo metodo risulta molto efficiente. Un algoritmo scritto in PASCAL, che esegue questo calcolo, è dato nella finestra 1. Qui facciamo solo un esempio: per calcolare $a^{149} \pmod n$, si calcolano a, a^2, a^4, a^8, a^{16}, a^{32}, a^{64} ed a^{128} per successive elevazioni al quadrato. In forma binaria, 149 si scrive come $(10010101)_2$, cioè $149 = 2^7 + 2^4 + 2^2 + 2^0$ e così si ottiene:

$$a^{149} \quad = \quad a \cdot a^4 \cdot a^{16} \cdot a^{128}.$$

Questo metodo per calcolare $a^m \pmod n$ richeide un numero di moltiplicazioni e di elevazioni al quadrato pari all'incirca al numero di cifre binarie dell'esponente m. Dunque si tratta di, al più, $\ln(m)$ moltiplicazioni modulo n, c ioè di una quantità di calcoli assai modesta.

```
function power(a, m, n: integer): integer;
(* the m-th power of a modulo n is evaluated;
n must be positive and m must be non-negative. *)
var x, y, z: integer;
begin x := a; y := 1; z := m;
  while z > 0 do
  begin if odd(z) then y := (y * x) mod n;
    x := (x * x) mod n; z := z div 2
  end;
  power:= y
end;
```

1. Elevare alla potenze m-esima modulo n.

Finalmente, due parole sulla divisione modulo n. Come tutti sanno, dividere per x è lo stesso che moltiplicare per l'inverso di x. Vediamo un esempio modulo 10:

$$3 \cdot 7 = 21 \equiv 1 \pmod{10},$$

e dunque 7 risulta l'inverso moltiplicativo di 3. In altre parole, modulo 10 vale la relazione

$$7 = \frac{1}{3} \pmod{10}.$$

In generale diremo che $x \in \mathbf{Z}$ ha un inverso moltiplicativo modulo n, se esiste un intero $y \in \mathbf{Z}$ tale che

$$x \cdot y \equiv 1 \pmod n.$$

Si vede facilmente che questa definizione è compatibile con l'identificazione dei numeri modulo n.

Non tutti gli interi ammettono un inverso moltiplicativo modulo n. Gli elementi di $\mathbf{Z}/n\mathbf{Z}$ che hanno inverso si dicono *le unità dell'anello* $\mathbf{Z}/n\mathbf{Z}$ e sono indicate con $(\mathbf{Z}/n\mathbf{Z})^*$. Si può dimostrare che un intero x ha inverso modulo n se e soltanto se $\mathrm{mcd}(x, n) = 1$ (qua mcd = massimo comun divisore). Per esempio, i soli numeri $x \in \mathbf{Z}$ che hanno inverso modulo 10 sono dati da

$$\{x \equiv 1,\ x \equiv 3,\ x \equiv 7,\ x \equiv 9 \pmod{10}\}.$$

In particolare, dal fatto che si ha

$$\begin{aligned} 3 \cdot 7 &\equiv 1 \pmod{10}, \\ 1 \cdot 1 \equiv 9 \cdot 9 &\equiv 1 \pmod{10}, \end{aligned}$$

segue che 3 e 7 sono uno l'inverso dell'altro, mentre 1 e 9 coincidono col proprio inverso. In conclusione, per decidere de un dato $x \in \mathbf{Z}$ ha inverso modulo n, basterà calcolare il massimo comun divisor fra x e n. L'algoritmo per fare questo calcolo è il cosiddetto *algoritmo di Euclide* ed è riportato in finestra 2.

Si tratta di un algoritmo molto efficiente, che si può considerare una degli algoritmi fondamentali per la teoria dei numeri.

```
function gcd(n, m: integer): integer;
(* the gcd of n and m is evaluated; *)
var x, y, q: integer;
begin x := abs(n); y := abs(m);
  while x > 0 do
  begin q := y mod x; y := x; x := q
  end;
  gcd:= y
end;
```

2. L'algoritmo di Euclide.

Come costruire grossi numeri primi.

Come sapeva già Euclide, i numeri primi sono infiniti. Essi sono anche assai densi fra i numeri interi: vicino ad un intero qualunque n, all'incirca uno, ogni $\ln(n)$ interi, è un numero primo. Ricordiamo che $\ln(n)$ è una quantità molto piccola rispetto ad n; ad esempio, se n è un numero di 100 cifre decimali, $\ln(n)$ è all'incirca uguale a 230. In altre parole, ci sono all'incirca 4000 primi che differiscono da n solo nelle ultime 6 cifre, mentre hanno in comune le prime 94. Questo è il contenuto del cosidetto *teorema dei numeri primi*, dimostrato alla fine del '800 da J. Hadamard e, indipendentemente, da C.J. de la Vallée-Poussin. Per ottenere numeri primi di grandezza assegnata, quello che faremo sarà di prendere a caso dei numeri di quella grandezza e poi di sottoporli a dei test di primalità. Per il teorema dei numeri primi dovremmo riuscire a trovare un numero primo n di μ cifre decimali con all'incirca $\ln(10) \cdot \mu$ tentativi.

Un ovvio test è il cosidetto *trial division algorithm*: dato l'intero n, si prova semplicemente a vedere se è divisibile per 2, 3, 4, 5, 6, ... etc. La cosa può essere sveltita per esempio provando solo 2 e gli interi dispari, come in finestra 3, oppure soltanto i numeri primi. Per essere sicuri che n sia primo, bisogna far vedere che non è divisibile per nessun intero minore o uguale alla sua radice quadrata (basta un attimo per convincersi che ogni numero composto n deve avere almeno un fattore minore di $\sqrt{n}$. Questo algoritmo è efficiente quando si tratta di numeri piccoli, ma diventa presto troppo dispendioso al crescere di n: già per un numero n di 20 cifre, bisogna provare almeno 1 000 000 000 di divisori. C'è da notare anche che il *trial division algorithm* è in realtà un algoritmo di fattorizzazione e, per quanto detto nell'introduzione, ciò sta ad indicare che non è il miglior test di primalità. Ad ogni modo, l'algoritmo è utile per trovare i divisori piccoli di un numero dato n.

```
function trialdivision(n, bound : integer): integer;
(* the output of trialdivision is the smallest divisor > 1 of n
that is less than min(bound, √n); if no such divisor is
found, the output is equal to 1 *)
var bd, m, e: integer;
begin m := abs(n); trialdivision := 1; e := 3; bd := bound;
  if bd * bd > m then bd := trunc(sqrt(m));
  if odd(m) then
  begin while e < bd do
    begin if (m mod e) = 0 then
      begin trialdivision := e; e := bd end;
      e := e + 2
    end
  end else
  if m > 2 then trialdivision := 2
end;
```

3. Il trial divison algorithm.

Come abbiamo spiegato nell'introduzione, esistono degli efficienti test di primalità in grado di trattare numeri fino a 100 cifre decimali, che è l'ordine di grandezza che ci interessa.

Questi test sono piuttosto involuti; ad ogni modo, per i nostri scopi, sarà sufficiente sapere che certi numeri sono "molto probabilmente" primi. Come vedremo siffatti numeri sono facili da costruire.

L'idea da cui nascono questi test di primnalità è quella di controllare se un dato numero n soddisfa i vari teoremi che sono soddisfatti da numeri primi. Il miglior esempio conosciuto è il *teorema di Fermat* (in piccolo):

Teorema. *(F) Se n è un numero primo, allora vale*

$$a^{n-1} \equiv 1 \pmod{n}, \qquad \textit{per ogni } a \in \mathbf{Z} \textit{ tale che } \mathrm{mcd}(a,n) = 1.$$

Come abbiamo visto, le potenze di a modulo m possono essere calcolate in modo efficiente e così la condizione contenuta nel teorema *(F)* può essere controllato velocemente. Sfortunatamente, non vale l'inverso di *(F)* e cioè esistono numeri che soddisfano *(F)* senza essere primi. Il più piccolo di essi è appunto $n = 561$. Ad ogni modo, c'è anche una versione raffinata del teorema *(F)*, che può essere di notevole utilità.

Sia n un intero positivo *dispari*. Scriviamo $n-1 = 2^k \cdot m$ dove m è dispari. Se n fosse primo, si verificherebbero allora i seguenti fatti. Sia $b \equiv a^m \pmod{n}$, ove a è un qualunque intero non divisibile per n; per il *teorema di Fermat (F)* si ha

$$b^{2^k} \quad \equiv \quad (x^m)^{2^k} \quad \equiv \quad x^{n-1} \quad \equiv \quad 1 \pmod{n}.$$

Sia ora i il più piccolo intero non negativo tale che

$$b^{2^i} \quad \equiv \quad 1 \pmod{n}.$$

Chiaramente abbiamo che $i \leq k$. Se vale $i > 0$, allora la potenza 'precedente' $c = b^{2^{i-1}}$ soddisfa la relazione

$$c^2 \quad \equiv \quad (b^{2^{i-1}})^2 \quad \equiv \quad b^{2^i} \quad \equiv \quad 1 \pmod{n}.$$

Dunque n divide $c^2 - 1 = (c-1)(c+1)$; per come è definito l'intero i, il primo n non divide $c-1$, cosicchè deve dividere $c+1$. In altre parole, abbiamo che $c \equiv -1 \pmod{n}$.

Ricapitolando:

$$(PS) \quad \begin{cases} \text{Se } n \text{ è primo e } b = a^m \pmod{n}, \text{ abbiamo che } b^{2^k} \equiv 1 \pmod{n}. \\ \text{Poi, se } i \text{ è il minimo intero non negativo tale che } b^{2^i} \equiv 1 \pmod{n}, \\ \text{allora ci sono due possibilità: abbiamo che } i = 0 \text{ (ossia } b \equiv \\ 1 \pmod{n}) \text{ oppure } b^{2^{i-1}} \equiv -1 \pmod{n}. \end{cases}$$

D'altra parte, se un numero n *non* è primo, si sa dimostrare che gli interi a che soddisfano le relazioni *(PS)*, costituiscono al più il 25% e. Per la maggior parte degli interi n, sono molti di meno. Sarà molto improbabile quindi che un numero non primi passi il seguente test:

sia n il numero in esame. Supponiamo che n sia dispari e scriviamo $n-1 = 2^k m$ per m dispari. Scegliamo a caso un intero a fra 0 e n e controlliamo che vale $\mathrm{mcd}(a, n) = 1$. *Calcoliamo $b \equiv a^m \pmod{n}$ e poi la successione b, b^2, b^4, ..., b^{2^k} come sopra e controlliamo che la relazione (PS) sia soddisfatta.*

Chiaramente, se fosse $\mathrm{mcd}(a, n) > 1$, si avrebbe già un fattore di n. I numeri che passano questo test si dicono *pseudoprimi* (rispetto ad a), oppure *a-pseudoprimi*. Se n risulta *pseudoprimo* per diversi valori di a, possiamo decisamente scommetere che è primo, pur non avendo una prova rigorosa. Ad ogni modo, per i nostri scopi, che sono i *criptosistemi*, è sufficiente lavorare con i numeri *pseudoprimi*.

```
function pseudoprime(n, a : integer): boolean;
(* pseudoprime returns 'true' if and only if n is a strong
pseudoprime with respect to the number a.
n must be odd and positive. *)
var b, k, m, c, i: integer;
begin m := abs(n) - 1; k := 0; i := 0; pseudoprime := true;
  while not odd(m) do begin m := m div 2; k := k + 1 end;
  b := power(a, m, n);
  if b <> 1 then
  begin while (b <> (n - 1)) and (i < k) do
    begin b := (b * b) mod n; i := i + 1 end;
    pseudoprime := (i < k)
  end
end
```

4. Test di pseudoprimalità.

La criptografia.

La criptografia è la scienza che si accupa della lettura di messaggi dati in forma cifrata, cosicché solo i destinatari siano in grado di decifrareli e di leggerli. Si tratta di una scienza molto antica, che si sviluppa specialmente in tempo di guerra.[1]

Vediamone un semplice esempio (pare che questo metodo risalga addirittura a Giulio Cesare): supponiamo che il testo originale sia scritto usando le 21 lettere dell'alfabeto $ABCDEFGHILMNOPQRSTUVZ$; supponiamo che, in questo ordine, le lettere sono numerate da 1 a 21. Il messaggio cifrato, o *criptotesto*, sarà ottenuto dal messaggio originale "traslando ciclicamente" tutte le lettere di un un numero fissato b. Per esempio, per $b = 3$, la A diventa D, la B diventa E, la V diventa B, le Z diventa C, ecc. In tal modo, ricevendo

$$DOHD \quad NDFZD \quad HVZ$$

e sapendo che $b = 3$, si può ricostruire il messaggio originale:

$$ALEA \quad IACTA \quad EST.$$

Come è facile intuire, questo metodo di cifrare i messaggi non 'è molto sicuro; dopo aver intercettato qualche messaggio, non sarà difficile indovinare la regola generale e quindi tradurre tutti i messaggi segreti.

Fino a pochi anni fa, tutti i metodi per trasmettere messaggi segreti avevano le seguenti caratteristiche:

- c'era un mutuo accordo segreto (per esempio, di "traslare" tutte le lettere di 3, come nel codice di Cesare);
- una volta riusciti a tradurre un certo numero di messaggi, si poteva quasi certamente risalire al codice e leggere tutti i messaggi successivi.

Oggigiorni i *criptosistemi*, ossia i sistemi per cifrare e decifrare i messaggi, sono usati in diversi ambienti oltre a quelli militari, come ad esempio nelle trasazioni finanziarie per via elettronica, nell'amministrazione di materiale riservato, ... ecc.[2] In queste circostanze sarebbe piuttosto sconveniente avere dei mutui accordi segreti fra ogni coppia di utenti. Così descriveremo ora un criptosistema moderno che funziona senza bisogno di tali accordi, ossia l'RSA, *criptosistema a chiave pubblica*, inventato da R.L. Rivest, A. Shamir e L.M. Adleman nel 1976. Al contrario, il modo in cui i messaggi vengono cifrati nell'RSA è pubblico! Chiunque voglia intrare nel sistema ha a sua disposizione tutti i messaggi decifrati che vuole. E nonostante ciò, il sistema sembra essere, in pratica, inpenetrabile!

Cominciamo con l'illustrare il sistema RSA con soli due utenti A e B, in seguito descriveremo il sistema completo RSA *a chiave pubblica* con nmuerosi utenti. Supponisma dunque che gli utenti A e B vogliano inviare dei messaggi cifrati. Anhce usando il sistema RSA, essi in realtà si invieranno dei numeri, in questo caso, numeri interi compresi fra

[1] Una lettura su questo argomento è il libro di David Kahn, *The code breakers, the story of secret writing*, 1967.

[2] Per un'introduzione a questi argomenti, rinviamo al libro di Neal Koblitz, *A course in number theory and cryptography*, Sprnger-Verlag 1987.

0 e$n-1$, over n è un certo intero fissato di circa 200 cifre decimali. Come nel caso del sistema di Cesare, è dunque necessario trasformare il testo del messaggio in una successione di numeri; questa volta si tratterà però di numeri di 200 cifre circa, compresi fra 0 e $n-1$. Ciò può essere fatto in diversi modi. Uno di essi consiste, ad esempio, nel dividere il testo in pezzi più piccoli, ognumo dei quali viene trasformato poi (usando il codice ASCII o qualcosa di simile) in numeri interi fra 0 e $n-1$. Tali numeri saranno anche chiamati "blocchi unitari". È da notare che questa procedura non ha niente a che fare con il *criptosistema* vero e proprio, ma è solo un preliminare.

Il criptosistema consiste poi nel manipolare i numeri così ottenuti. Nel caso del sistema di Cesare si trattava di traslare tutto di 3. Nel caso del sistema RSA, come vedremo, la cosa è assai più complicata. Nel sistema RSA si prende un intero n, prodotto di sue numeri primi p e q di circa 100 cifre ciascuno[3]. Nella pratica, i primi p e q sono scelto in modo da soddisfare anche altre proprietà, su cui però non vale la pena soffermarsi. Per p e q vale la ben nota generalizzazione del *teorema di Fermat* (in piccolo):

$$a^{(p-1)(q-1)} \equiv 1 \pmod{n}, \qquad \text{per ogni } a \in \mathbf{Z} \text{ tale che } \mathrm{mcd}(a,n) = 1.$$

Viene poi scelto un numero a caso $E \in \mathbf{Z}$ con la proprietà che

$$\mathrm{mcd}(E, (p-1)(q-1)) \;=\; 1,$$

in modo da assicurare l'esistenza di un inverso moltiplicativo di E *modulo* $(p-1)(q-1)$ (come si è visto nel paragrafo sul calcolo modulo n). In altre parole, esisterà un intero D tale che

$$D \cdot E \equiv 1 \pmod{(p-1)(q-1)}$$

ed esso potrà calcolato esplicitamente sfruttando *l'algoritmo di Euclide*, mentre si fa il calcolo verificando che $\mathrm{mcd}(E,(p-1)(q-1))$ è uguale a 1. In finestra 5 è riportata questa estensione dell'algoritmo di Euclide.

```
function egcd(e, n : integer; var b : integer): boolean;
(* b e egcd will be computed such that a * b is congruent to egcd
modulo n.
n must be positive *)
var x, y, z, m, q, r: integer;
begin m := abs(n); x := a mod n; y := 0; z := 1;
  while  x <> 0 do
  begin q := m div x;
    r := x; x := m - q * x; m := r;
    r := z; z := y - q * z; y := r
  end
  b := y; egcd := m
end
```

[3] In pratica, ciò presuppone la disponibilità di un package di aritmetica intera estesa (a precisione multipla/estesa)

Ora, mentre i numeri n ed E sono pubblici, la fattorizzazione di n e il numero D sono tenuti segreti dagli utenti.

Com'è che i messaggi vengono cifrati? Come abbiamo spiegato, un messaggio consiste di "blocchi unitari", che sono interi fra 0 e n. La probabilità che per uno di questi numeri a sia $\mathrm{mcd}(a,n) > 1$ è di circa 10^{-100}; noi dunque ignoreremo questa possibilità e assumeremo $\mathrm{mcd}(a,n) = 1$. L'utente A cifra il messaggio A, elevenadolo alla potenza E *modulo* n ed invia a B

$$a^E \pmod{n},$$

compreso fra 0 e $n-1$. Il ricevente B a sua volta, eleva il testo cifrato alla potenza segreta potenza D modulo n. Ottiene in tal modo

$$(a^E)^D \;\equiv\; a^{ED-1} \cdot a \;\equiv\; 1 \cdot a \;\equiv\; a \pmod{n}$$

poichè la differenza $ED-1$ è un multiplo di $(p-1)(q-1)$ e si può applicare il teorema di Fermat generalizzato, menzionato sopra.

Ora come ora, sembra che il solo modo in cui un terzo possa "rompere" il sistema, sia quello di venire in poessesso di D. Conoscendo la fottorizzazione di $n = p \cdot q$, è molto facile determinare un intero D tale che $E \cdot D$ è congruo a 1 (mod $(p-1(q-1))$, e dicifrare poi tutti i messaggi. Anche se il numero n e la chiave E sono resi pubblici, sembra inimmaginabile che qualcuno possa calcolarsi un numero D tale che $a^{ED} \equiv a \pmod{n}$, *senza* conoscere il numero $(p-1)(q-1)$. Notiamo che passare da $t = (p-1)(q-1)$ all fattorizzazione di $n = p \cdot q$ equivale semplicemente a risolvere l'equazione di secondo grado $X^2 + (n-t+1)X + n = 0$. In conclusione, "rompere" il sistema sembra equivalente a fattorizzare n e la sicurezza del sistema si basa sul fatto che fino ad ora nessuno è in grado di fattorizzare in modo efficiente numeri di 200 cifre. Per il resto, installare un tale sistema è relativamente semplice: come abbiamo visto, ci sono metodi efficienti per generare numeri primi (o *pseudoprimi*) di 100 cifre decimali (vedi finestra 4): cifrare e decifrare consiste poi nel calcolo di potenze di un numero modulo n, ed anche questo può essere fatto in modo efficiente (vedi finestra 1).

Concludiamo questo paragrafo discutendo il *criptosistema RSA a chiave pubblica*, con più di due utenti. Questa volta però ogni utenti u renderà pubblico un numero n_u e una chiave E_u per cifrare i suoi messaggi. Come in precedenza, tutti i messaggi per l'utente u verrano inviati in "blocchi unitari" che sono interi fra 0 e n_u. Ogni utente u terrà invece segreta la chiave D_u per la decifrazione. Le chiavi E_u e D_u soddisfano la relazione

$$a^{E_u D_u} \equiv 1 \pmod{n_u}, \qquad \text{per ogni } a \in \mathbf{Z} \text{ tale che } \mathrm{mcd}(a, n_u) = 1.$$

Anche questa volta ogni numero n_u deve essere il prodotto di due primi di circa 100 cifre l'uno, che sono però tenuti segreti. I numeri n_u e E_u saranno pubblicati su un elenco accanto al nome dell'utente, così da poter essere consultati dagli altri. Quando infatti l'utente A vuol mandare un messaggio all'utente B, egli andrà a consultare l'elenco per conoscere i numeri n_B ed E_B di B. Invierà poi a B:

$$a^{E_B} \pmod{n_B};$$

l'utente B a sua volta decifrerà il messaggio di A calcolando, per mezzo della sua chiave segreta D_E

$$(a^{E_B})^{D_B} \equiv a \pmod{n_B}.$$

In questo modo, numerois utenti hanno accesso al sistema, senza bisogno di accordi individuali segreti.

Le firme.

A volte può essere importante sapere con certezza chi ha mandato un certo messaggio. Tradizionalmente, è la firma del mandante a servire questo scopo. Nelle comunicazioni elettroniche, è necessario basarsi su altri metodi. Per esempio, quando un impiegato di una ditta vuole prelevare dei soldi dal conto della ditta per telefono, aprova delle sua identità, gli vengono richiesti dei dati, che difficilmente altri possono sapere. Tali dati, come ad esempio un numero o il nome di un parente erano stati "depositati" alla banca al momento dell'apertura del conto.

Il *criptosistema RSA a chiave pubblica* (come altri criptosistemi del genere) offre un metodo più sicuro per autenticare i messaggi; l'utente A aggiunge al suo messaggio, inviato all'utente B, qualcosa della forma $S^{D_A} \pmod{n}$, ove S è una sorte di firma, come il nome o la data di spedizione. A questo punto l'intero messaggio viene cifrato elevando i "blocchi unitari" a alla potenza E_B (modulo n). Qundo l'utente B riceve il messaggio cifrato e lo decifra usando la sua chiave segreta D_E, troverà all fine una riga incomprensibile e precisamente S^{D_A}. Sapendo che il messaggio dovrebbe provenire da Am l'utente B va a controllare sull'elenco degli utenti il numero E_A di A e poi calcola

$$(S^{D_A})^{E_A} \pmod{n_A}$$

trovando S, ossia riuscendo a decifrare la firma, B può essere sicuro che a mandare il messaggio era effettivamente A. Contraffare una firma è difficile quanto "rompere" il criptosistema, che a sua volta equivale a fattorizzare numeri interi di 200 cifre.

Un'altra applicazione di un criptosistema a chiave pubblica è la seguente, preso dal libro di Neal Koblitz. Supponiamoc eh due paesi A e B abbiano raggiunto un accordo sui test nucleari sotterranei e che debbono verificare il rispetto del trattato. A questo scopo i due paesi si dotano di sue sismografi identici, che trasmettono via radio le loro registrazioni. Un sistema in grado di prevenire contraffazioni dei dati deve avere le seguenti caratteritsiche:

(a) il paese A deve sapere il contenuto dei messaggi che partono dal suo territorio, per assicurarsi che l'apparecchio non sia usato dal paese B per scopi di spionaggio (e viceversa);
(b) il paese A deve assicurarsi che il pase B non fabbrichi messaggi falsi, ad esempio non mandi a dire che è tutto calmo, mentre invece procede negli esperimenti nucleari (e viceversa);
(c) un qualunque terzo deve essere in grado in caso di disputa di leggere i messaggi provenienti da ambo le parti.

Tutte queste esigenze sono soddisfatte usando un criptosistema a chiave pubblica: il sismografo sarà attacato ad un computer che generi a caso i numeri n e le chiavi D per la

decodifica; la chiave di *codifica* resta invece segreta. In questo modo, ognuno può leggere i messaggi cifrati trasmessi dai sismografi, ma nessun può contraffarli.