Aritmetica e crittografia: l'algoritmo RSA

Michele Impedovo

Riassunto. Questo articolo è il resoconto di un'esperienza di lavoro svoltasi il 22 aprile e il 6 maggio 2006 presso il Liceo Scientifico A. Issel di Finale Ligure, nell'ambito di un progetto coordinato dal prof. Domingo Paola. L'obiettivo è stato fornire le conoscenze matematiche necessarie al fine di descrivere in modo dettagliato il metodo RSA per la criptazione dei messaggi. Parallelamente sono stati implementati con Derive gli algoritmi necessari e ne è stata analizzata e testata la loro calcolabilità. È stato così possibile concludere l'esperienza con una prova effettiva di codifica-decodifica di messaggi.

Abstract. This paper is the report of an experience carried out at Liceo Scientifico Issel of Finale Ligure, within a plan coordinated from Domingo Paola. The aim was to provide with mathematical knowledge in order to describe in detail the RSA method for encryption and decryption of messages. Meanwhile we implemented with Derive the necessary algorithms and we actually tested them. It has therefore been possible to conclude the experience with an effective test of encryption-decryption of messages.

Michele Impedovo
Università Bocconi di Milano
michele.impedovo@uni-bocconi.it
www.matematica.it/impedovo

Introduzione

Questo articolo è il resoconto di un'esperienza di lavoro con alcuni docenti e alcuni studenti di classe III e IV del Liceo Scientifico A. Issel di Finale Ligure. Il lavoro si è articolato in due mattine, il 22 aprile e il 6 maggio 2006. È stata una delle tante iniziative che hanno costituito il progetto "*Il numero nella vita quotidiana: come accrescere l'interesse degli studenti verso la matematica*", coordinato dal prof. Domingo Paola. Questo progetto ha vinto il concorso *Centoscuole* (in palio 15.000 euro) promosso dalla Fondazione per la Scuola (www.fondazionescuola.it).

Il tema degli incontri è stato la **crittografia a chiave pubblica**, e in particolare l'algoritmo **RSA**. Per progettare un percorso realizzabile e soprattutto sperimentabile dagli studenti, ho dovuto procedere a ritroso, dall'algoritmo RSA al Teorema di Eulero e al Piccolo Teorema di Fermat, fino alle operazioni di addizione e moltiplicazione in $\mathbf{Z}_n$ e al codice ASCII.

Durante le due lezioni, a mano a mano che si procedeva, si implementavano gli algoritmi, dapprima con la TI-89 e poi, quando abbiamo avuto bisogno di potenza di calcolo, con Derive.

Tutti i file descritti nell'articolo (e molti altri) sono contenuti nei due file

www.matematica.it/impedovo/articoli/derive.zip

www.matematica.it/impedovo/articoli/ti89.zip

Abbiamo deciso, per semplicità, di utilizzare un alfabeto ridotto a 27 caratteri: le 26 lettere minuscole e lo spazio; infatti utilizzare i 128 codici ASCII (o i 256 codici dell'ASCII esteso) comporta non solo un rallentamento dal punto di vista computazionale, ma anche fastidiose difficoltà nella gestione dei caratteri di controllo (TAB, ENTER, ESC, …).

Al termine del secondo incontro gli studenti, con una certa soddisfazione, si scambiavano messaggi criptati da un computer all'altro.

Per la stesura di questo articolo devo ringraziare Domingo Paola, che mi ha permesso di vivere questa bellissima esperienza didattica, e che ha insistito a lungo perché la mettessi nero su bianco.

La crittografia a chiave segreta

Il primo codice crittografico di cui si ha testimonianza storica è il semplicissimo **codice di Cesare** (Svetonio, *Vitae Caesarum*, I, 56); consiste nel sostituire ad ogni carattere C di un messaggio un altro carattere C' che si trova, nell'ordine alfabetico, *n* posti più avanti rispetto a C; se si supera la "*z*" si ricomincia a contare dalla "*a*".

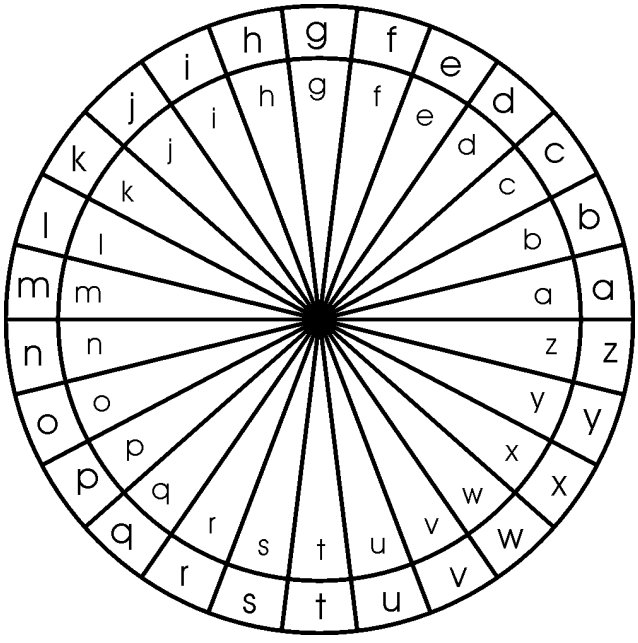
Per esempio, se $n = 7$, allora il messaggio

"*matematica*"

viene codificato in

"*thalthapjh*".

Chi riceve il messaggio associa ad ogni carattere la lettera che si trova 7 posti più indietro (se si supera la "*a*" si ricomincia a contare dalla "*z*") e lo decodifica.



L'immagine precedente suggerisce uno strumento (due dischi sovrapposti che ruotano uno sull'altro) che si potrebbe costruire alla scuola elementare per realizzare un gioco-attività con lettere e numeri sul codice di Cesare.

Il numero *n* è la **chiave** dell'algoritmo, la cosiddetta **chiave segreta**: non deve essere nota ad altri e deve essere preliminarmente concordata dai due utenti.

Il codice di Cesare è facilmente scardinabile: sia perché le possibili diverse chiavi sono solo 26 (basta provarle tutte), sia perché in ogni lingua le lettere compaiono con una certa frequenza statistica (vedi ad esempio www.riksoft.com/criafreq.htm) e dunque, soprattutto se il messaggio è abbastanza lungo, non è difficile formulare congetture sensate e verificarle con rapidità. Nel nostro esempio è naturale osservare che nella parola *thalthapjh* ci sono tre "*h*", e che con tutta probabilità rappresentano una vocale; con pochi tentativi si ottiene la decodifica.

Inoltre la chiave *n*, per essere affidabile, dovrebbe essere modificata spesso, con ovvie difficoltà di comunicazione tra i due utenti.

Il **codice di Vigenère** (Blaise de Vigenère, 1523-1596) introduce una variante che lo rende decisamente più sicuro del codice di Cesare: le lettere non sono traslate tutte dello stesso valore *n*. Più precisamente: i due utenti si scambiano preliminarmente un vettore numerico di lunghezza arbitraria, tipicamente la sequenza numerica corrispondente ad una parola; per esempio [1,13,15,18,5] corrisponde alla parola "*amore*" ed è la chiave segreta dell'algoritmo. La codifica del messaggio avviene nel seguente modo:

- il 1° carattere viene traslato di 1 posto;
- il 2° carattere viene traslato di 13 posti;
- il 3° carattere viene traslato di 15 posti;
- il 4° carattere viene traslato di 18 posti
- il 5° carattere viene traslato di 5 posti;
- il 6° carattere viene traslato di 1 posto;
- il 7° carattere viene traslato di 13 posti
- …

e così via. La parola "*matematica*" viene così trasformata in

"*nniwrbgxuf*".

Lettere uguali nel messaggio originale non corrispondono in generale a lettere uguali nel messaggio cifrato (e viceversa), e questo lo mette al riparo da una decodifica immediata; ovviamente, tanto più lunga è la chiave segreta, tanto più sicuro è il codice. Tuttavia non è difficile immaginare che, per esempio utilizzando un calcolatore, si possa in breve tempo scoprire la chiave segreta

e decodificare il messaggio. Un metodo efficiente per scardinare il codice di Vigenère è stato proposto, ben prima dell'avvento dei calcolatori, da Friedrich Kasiski (1805-1881); vedi per esempio

http://www.trincoll.edu/depts/cpsc/cryptography/vigenere.html.

La rivoluzione della chiave pubblica

Gli algoritmi di Cesare e di Vigenère hanno la stessa struttura: ad una stringa S il mittente applica una certa funzione f e invia al destinatario il messaggio criptato $f(S)$. Il destinatario applica a $f(S)$ la funzione inversa f^{-1} e legge il messaggio in chiaro $f^{-1}(f(S)) = S$. In entrambi gli algoritmi la segretezza del messaggio è affidata alla segretezza della chiave; nota questa, la funzione inversa è immediatamente disponibile: se la chiave di codifica è il numero n, o il vettore $\mathbf{v}$, quella di decodifica è il numero ^{-}n, o il vettore $^{-}\mathbf{v}$.

La crittografia a chiave pubblica ha rivoluzionato i meccanismi di codifica e decodifica dei messaggi: non occorre più che mittente e destinatario si mettano preliminarmente d'accordo su quale chiave segreta utilizzare.

Nell'epoca di Internet e delle transazioni elettroniche, è indispensabile che B possa mandare un messaggio segreto ad A (per esempio il numero della propria carta di credito), senza dover preliminarmente comunicare ad A quale chiave segreta utilizzare. Infatti tale comunicazione viaggerebbe anch'essa su canali esposti ad eventuali malintenzionati. Se C intercetta la chiave segreta, può facilmente decodificare qualunque messaggio.

Gli algoritmi a chiave pubblica eliminano questa falla: ad ogni utente vengono assegnate, da una Certification Authority (spesso abbreviata in CA, vedi per esempio www.thawte.com), una **chiave pubblica**, che compare in un elenco consultabile da chiunque, e una **chiave privata**, che l'utente deve tenere gelosamente segreta.

Se B vuole mandare un messaggio ad A, non deve prima concordare con A quale chiave usare; deve semplicemente cercare sull'elenco la chiave pubblica di A, e cifrare il messaggio, mediante un opportuno algoritmo di pubblico dominio, con questa chiave. Una volta cifrato, il messaggio può essere decodificato solo da chi conosce la chiave privata, e dunque solo da A. Perché questo sia possibile, è necessario che la funzione f (che codifica il messaggio) e la funzione f^{-1} (che lo decodifica) non siano simmetriche: cioè non deve essere possibile ricavare f^{-1} direttamente da f.

Si obietterà: ma se la funzione f è pubblica, non dovrebbero esserci problemi particolari a ricavare f^{-1}, soprattutto nell'era della rivoluzione elettronica. Ci si chiede: è possibile progettare un algoritmo per il quale sia praticamente impossibile ricavare f^{-1} da f? La metafora dell'**elenco telefonico** può servire a capire che un tale algoritmo, almeno in linea di principio, è possibile. Trovare il numero di telefono di un utente a partire dal nome è immediato (questa è la funzione f); invece trovare il nome di un utente partendo dal suo numero di telefono (la funzione inversa f^{-1}) è praticamente impossibile, perché richiederebbe una quantità di tempo tale da risultare nei fatti impraticabile. Nel seguito utilizzeremo spesso il termine "impossibile" in questo senso: non calcolabile in tempi ragionevoli.

Il primo algoritmo che soddisfa i requisiti della chiave pubblica è stato messo a punto solo nel 1977 da Ron **R**ivest, Adli **S**hamir e Leonard **A**dlemann, ed è noto, dalle iniziali degli autori, come **algoritmo RSA**. Ancor oggi è uno dei più utilizzati: vedi per esempio www.rsasecutity.com.

Come vedremo in dettaglio, per l'algoritmo RSA la funzione "facile da calcolare" è la **moltiplicazione** di due numeri primi p e q molto grandi; oggi l'ordine di grandezza di assoluta sicurezza utilizzato per p e q è circa 10^{300}: noti p e q, il prodotto $n = pq$ si calcola immediatamente. La funzione inversa, quella "impossibile da calcolare", è la **fattorizzazione**: se un numero n è il prodotto di due numeri primi "grandi", la sua fattorizzazione richiede di norma tempi superiori all'età dell'universo. Infatti non si conoscono a tutt'oggi (e c'è ragione di ritenere che non si conosceranno mai), algoritmi efficienti di fattorizzazione.

È interessante notare, a questo proposito, l'enorme sproporzione degli attuali risultati della ricerca matematica avanzata sui due problemi apparentemente analoghi:

1) stabilire se un numero naturale è primo;
2) fattorizzare un numero naturale.

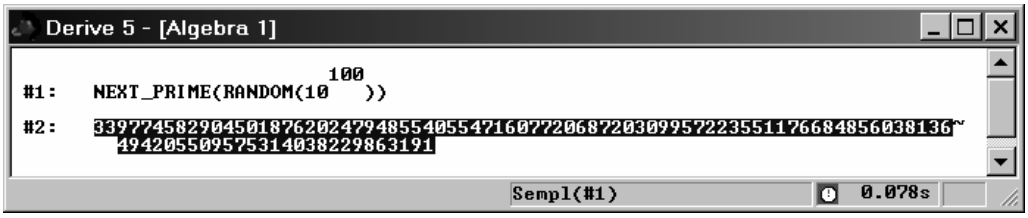
Il primo problema è "facile", il secondo è "impossibile".

Il riconoscimento (e quindi la costruzione) di numeri primi grandi non offre oggi particolari difficoltà computazionali. Esistono algoritmi molto efficienti in grado di riconoscere numeri primi anche di migliaia di cifre in pochi decimi di secondo.

I più efficienti di essi (vedi per esempio l'algoritmo di Miller-Rabin, oppure l'algoritmo di Solovay-Strassen, che viene illustrato nell'Appendice C) sono di tipo probabilistico; viene effettuato un certo numero di test sul numero e:

- se tutti i test vengono superati, il numero viene dichiarato primo con una probabilità che può essere resa vicina a 1 quanto si vuole in tempi rapidi;
- se uno solo dei test non viene superato allora il numero è certamente composto.

Con Derive il comando `NEXT_PRIME(n)` determina il più piccolo numero primo maggiore di *n*. La figura seguente mostra come venga determinato il *next prime* di un numero casuale di 100 cifre in 0.078 secondi (i tempi di calcolo sono in basso a destra).



Il più grande numero primo fino ad oggi conosciuto (giugno 2006) è il 43° **numero di Mersenne** (cioè della forma 2^p-1):

$$M_{43} = 2^{30\,402\,457}-1.$$

Si tratta di un numero che ha quasi 10 milioni di cifre, per l'esattezza

9 152 052 cifre

(vedi http://primes.utm.edu).

Per quanto riguarda invece la fattorizzazione del prodotto di due numeri primi (vedi per esempio http://mathworld.wolfram.com/news/2005-11-08/rsa-640/), è del maggio 2005 il lavoro di un gruppo di ricerca dell'Università di Bonn che, dopo 5 mesi di attività supportata da 80 CPU da 2.2 GHz connesse in rete, ha condotto a fattorizzare RSA-640, un numero di "soltanto" 193 cifre. Questa sfida è sponsorizzata proprio dalla RSA Security: la prossima sfida ancora aperta riguarda RSA-704, un numero di 212 cifre, e ci sono in palio 30000 dollari per chi riesca a fattorizzarlo.

Si può utilizzare un qualunque software di matematica per rendersi conto, almeno in modo empirico e del tutto qualitativo, dei tempi necessari per la fattorizzazione di un numero naturale *n* in funzione della lunghezza di *n*, cioè del numero di cifre di *n*.

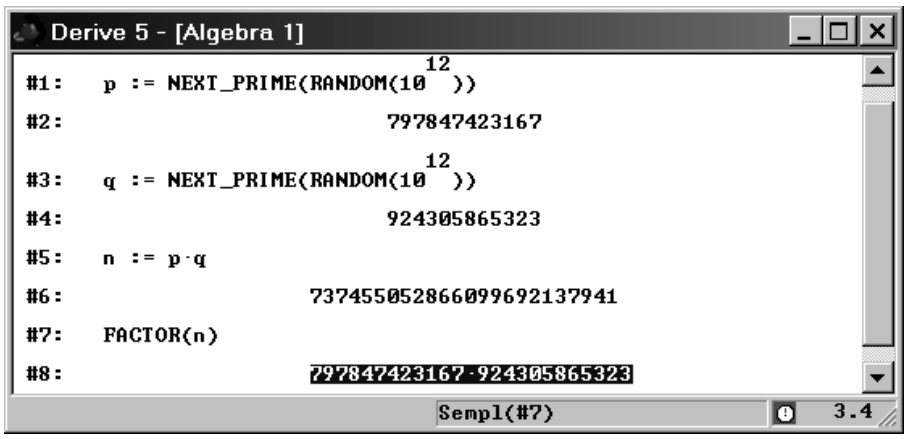
Abbiamo provato, con Derive, a rilevare i tempi per la fattorizzazione di numeri $n = pq$ ottenuti moltiplicando due numeri primi scelti casualmente con

il comando

NEXT_PRIME(RANDOM(10^k)),

con $k = 8, 10, 12, 14, 16$; così p e q hanno ordine di grandezza da 10^8 a 10^{16} e n ha ordine di grandezza da 10^{16} a 10^{32}.

Per ogni valore di k abbiamo eseguito 10 esperimenti e ne abbiamo preso la media. La figura seguente mostra un esperimento con $k = 12$ e $n \approx 10^{24}$; in basso a destra si nota il tempo di fattorizzazione.



La tabella dell'intero esperimento è la seguente.

cifre di *n*	16	20	24	28	32	36
tempo (s)	0.05	0.36	1.78	15.6	103	673

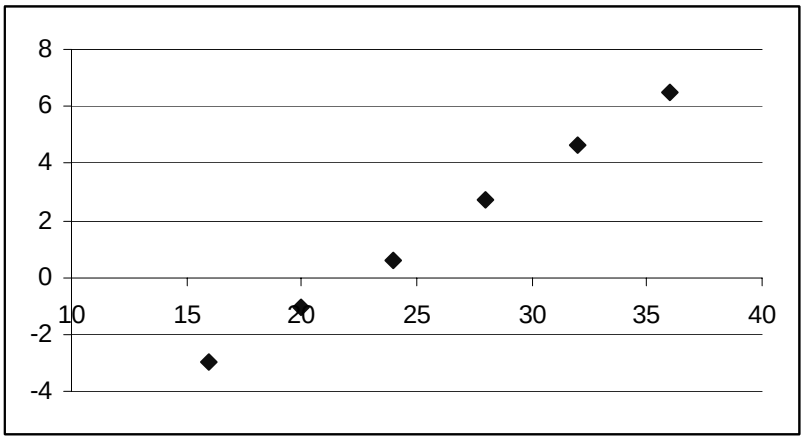
Supponiamo che i tempi T siano esponenziali rispetto alla lunghezza G di n:

$$T = ae^{bG}.$$

Allora risulterebbe

$$\ln(T) = \ln(a)+bG,$$

cioè il logaritmo dei tempi sarebbe lineare, di pendenza b e intercetta $\ln(a)$, rispetto alla lunghezza G di n. Il grafico di $\ln(T)$ rispetto a G è il seguente.



La retta di regressione dei punti risulta essere circa

$$y = 0.48x - 10.6$$

da cui ricaviamo una stima grossolana della funzione che lega T a G:

$$T \approx 0.000024 \cdot e^{0.48G}.$$

Se estrapoliamo i dati da questa stima concludiamo che per fattorizzare numeri di 50 cifre occorrerebbe a Derive circa una settimana e per numeri di 100 cifre circa 500 milioni di anni.

Con i più potenti algoritmi e con i più potenti calcolatori disponibili oggi, cambierebbero i dati, ma non la sostanza: i tempi di fattorizzazione restano esponenziali rispetto alla lunghezza di n. La fattorizzazione del prodotto di due numeri primi di 300 cifre richiede tempi superiori all'età dell'universo.

L'algoritmo RSA funziona perché a tutt'oggi non si conoscono algoritmi efficienti di fattorizzazione (nonostante la ricerca matematica sia attivissima in questo settore), ma non è stato dimostrato da alcuno che sia inattaccabile. RSA perderebbe istantaneamente la sua inattaccabilità il giorno in cui si scoprisse un algoritmo di fattorizzazione capace di lavorare in tempi polinomiali (e non esponenziali) rispetto alla lunghezza dell'input.

Dato che RSA si basa sulla non esistenza di algoritmi rapidi di fattorizzazione, nella trattazione che segue dovremo sempre porre attenzione non solo agli aspetti matematici, ma anche a quelli computazionali. Non avrebbe senso illustrare un procedimento senza preoccuparci, passo-passo, della sua effettiva calcolabilità. Voglio precisare che l'attenzione agli aspetti computazionali sarà del tutto qualitativa: ci si limiterà a verificare in modo empirico i tempi di calcolo degli algoritmi.

Noi lavoreremo con Derive e utilizzeremo fattori p e q di 300 cifre, perciò la lunghezza di n sarà di 600 cifre. Quindi lavoreremo con numeri il cui ordine di grandezza è quello effettivamente utilizzato nella pratica: vogliamo mostrare che anche un software non professionale come Derive può essere adatto allo scopo.

Lo schema generale

Vediamo lo schema generale adottato da un codice crittografico; se l'utente B deve mandare ad A un testo T, cioè una stringa di caratteri, allora:

1) B trasforma T in un numero naturale N, mediante una funzione g nota a tutti gli utenti;
2) B applica al numero N la funzione di codifica f, nota a tutti, ottenendo un numero M;
3) B ritrasforma il numero M in un testo T' (è il testo cifrato) mediante la funzione g^{-1}, inversa di g e nota a tutti, e manda T' ad A:

$$T \xrightarrow{g} N \xrightarrow{f} M \xrightarrow{g^{-1}} T'$$

4) A riceve T' e lo trasforma in M mediante la funzione g;
5) A applica al numero M la funzione di decodifica f^{-1}, nota solo ad A, e ottiene il numero N;
6) A applica a N la funzione g^{-1} e ottiene finalmente il messaggio originale:

$$T' \xrightarrow{g} M \xrightarrow{f^{-1}} N \xrightarrow{g^{-1}} T$$

I passaggi 3 e 4 non sono ovviamente necessari (B potrebbe consegnare ad A direttamente il numero M), ma hanno solo una valenza estetica: parte una stringa di testo, arriva una stringa di testo.

Ci occuperemo inizialmente delle funzioni g e g^{-1} che trasformano un testo in un numero naturale e viceversa, e poi delle due funzioni aritmetiche f e f^{-1} che costituiscono il cuore dell'algoritmo crittografico.

Dal testo al numero e viceversa

Il punto di partenza per trasformare caratteri in numeri è il codice ASCII (American Standard Code for International Interchange), che definisce una corrispondenza biunivoca tra i 128 (o 256 per l'ASCII esteso) caratteri dello standard ISO e i numeri naturali 0, 1, …, 127 (o 0, 1, …, 255).

Con Excel è possibile ottenere il carattere corrispondente al codice *n* mediante la funzione

`=codice(n)`

e viceversa, il codice ASCII di un carattere *c* mediante la funzione

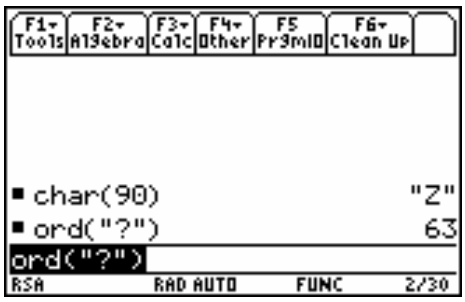
`=codice.caratt("c")`.

Le due funzioni sono, ovviamente, una inversa dell'altra. Ecco i caratteri corrispondenti ai codici da 32 a 127.

32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55
	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/	0	1	2	3	4	5	6	7
56	**57**	**58**	**59**	**60**	**61**	**62**	**63**	**64**	**65**	**66**	**67**	**68**	**69**	**70**	**71**	**72**	**73**	**74**	**75**	**76**	**77**	**78**	**79**
8	9	:	;	<	=	>	?	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
80	**81**	**82**	**83**	**84**	**85**	**86**	**87**	**88**	**89**	**90**	**91**	**92**	**93**	**94**	**95**	**96**	**97**	**98**	**99**	**100**	**101**	**102**	**103**
P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_	`	a	b	c	d	e	f	g
104	**105**	**106**	**107**	**108**	**109**	**110**	**111**	**112**	**113**	**114**	**115**	**116**	**117**	**118**	**119**	**120**	**121**	**122**	**123**	**124**	**125**	**126**	**127**
h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	

I primi 32 caratteri (codici ASCII da 0 a 31) sono caratteri di controllo e non sono stampabili; il codice 32 è lo spazio " "; da 48 a 57 abbiamo le cifre, da 65 a 90 le 26 lettere maiuscole, da 97 a 122 le 26 lettere minuscole. I caratteri da 128 a 255 comprendono lettere accentate e altri simboli.

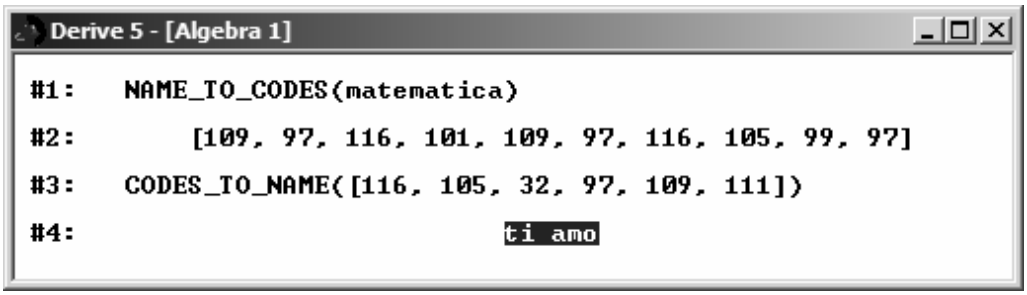
Sulla TI-89 le corrispondenti funzioni sono `char(n)` e `ord("c")`.



Derive tratta le stringhe come vettori; la funzione

`name_to_codes(s)`

trasforma la stringa "s" nel vettore **v** i cui elementi sono i codici ASCII dei caratteri di s. La funzione inversa è `codes_to_name(v)`.



Per trasformare un testo in un numero naturale, utilizziamo per semplicità l'alfabeto ridotto

{" ", "a", "b", …, "z"}

e mettiamolo in corrispondenza biunivoca con i numeri

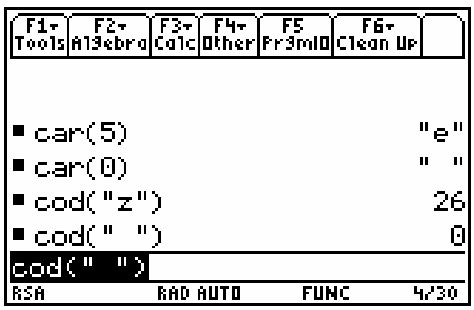
{0, 1, 2, …, 26},

in modo da poter lavorare, come vedremo, in $\mathbf{Z}_{27}$.

Con la TI-89, per costruire questa corrispondenza biunivoca, definiamo le funzioni `cod(c)` e `car(n)`, che sono una l'inversa dell'altra.

```
when(c=" ",0,ord(c)-96)→cod(c)
when(n=0," ",char(n+96))→car(n)
```

La funzione `cod(c)` restituisce il numero di $\mathbf{Z}_{27}$ corrispondente al carattere "c", la funzione `car(n)` restituisce il carattere dell'alfabeto corrispondente al numero *n* di $\mathbf{Z}_{27}$.



Ora passiamo dal singolo carattere alla stringa; costruiamo la funzione g (la chiameremo "numero") e la sua inversa g^{-1} (la chiameremo "testo"). A questo scopo è sufficiente esprimere un numero in base 27. Per esempio, alla stringa "abc" associamo il numero 123 in base 27, cioè il numero (in base 10)

$$1\cdot 27^2 + 2\cdot 27 + 3 = 786.$$

Un algoritmo iterativo per trasformare una stringa in un numero naturale consiste nel partire da $n = 0$, e ad ogni passo moltiplicare per 27 e aggiungere la cifra successiva:

$n = 0$
$n = 0\cdot 27 + 1 = 1$
$n = 1\cdot 27 + 2 = 29$
$n = 29\cdot 27+3 = 786$

Per tornare indietro, dal numero alla stringa, si parte dal numero 786 e ad ogni passo si calcola il resto rispetto a 27, lo si sottrae e si divide per 27:

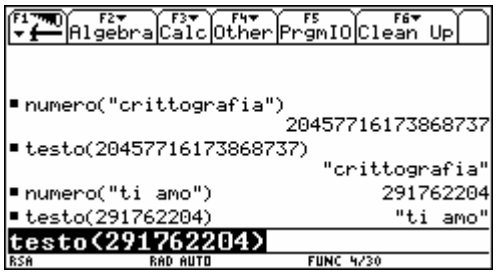
786	mod(786, 27) = **3**
(786–3)/27 = 29	mod(29, 27) = **2**
(29–1)/27 = 1	mod(1, 27) = **1**
(1–1)/27 = 0	

Ecco le due funzioni `numero` e `testo`, cioè g e g^{-1}, con la TI-89.

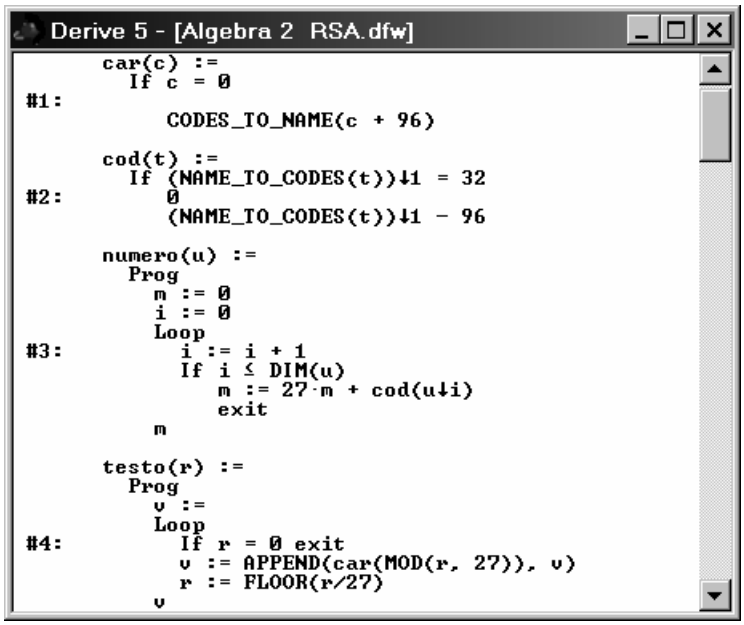
```
numero(s)
Func
Local n,i
0→n
For i,1,dim(s)
n*27+cod(mid(s,i,1))→n
EndFor
n
EndFunc
```

```
testo(n)
Func
Local s,r
""→s
While n>0
mod(n,27)→r
car(r)&s→s
(n-r)/27→n
EndWhile
s
EndFunc
```

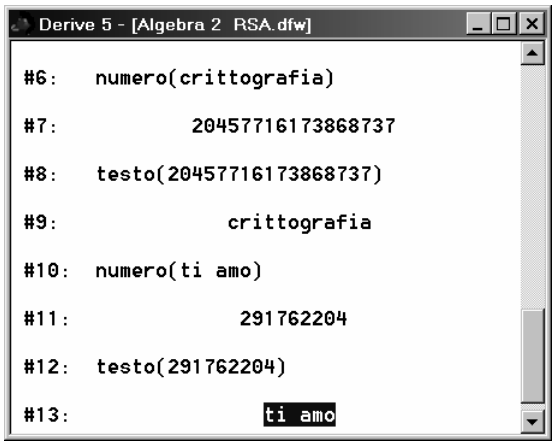
Ed ecco le due funzioni al lavoro.



Con Derive la sintassi delle funzioni `car`, `cod`, `numero`, `testo`, è lievemente differente (la programmazione è comunque più faticosa con Derive, perché il programma va scritto in linea). La figura seguente mostra l'implementazione delle quattro funzioni, ottenuta utilizzando opportunamente i comandi `NAME_TO_CODES` e `CODES_TO_NAME`.



Nella figura seguente sono mostrati gli esempi già svolti con la TI-89, con gli stessi risultati.



Ora siamo in grado di trasformare un testo in un numero e viceversa. È giunto il momento di introdurre il bagaglio matematico che ci servirà per definire l'algoritmo RSA.

Il Teorema di Fermat

L'aritmetica che serve per implementare l'algoritmo RSA è quella della struttura algebrica $\mathbf{Z}_n$: l'insieme delle **classi di resto modulo *n***, cioè l'insieme $\{0, 1, \ldots, n-1\}$ in cui sono definite la somma e il prodotto nel seguente modo:

- $a+b = \mathrm{mod}(a+b, n)$
- $a \cdot b = \mathrm{mod}(a \cdot b, n)$

dove `mod(x,y)` è la funzione (implementata in qualsiasi sistema di calcolo) che restituisce il **resto** del quoziente tra numeri interi x/y.

La figura seguente illustra le tavole pitagoriche della somma e del prodotto in $\mathbf{Z}_9$ (nella tabella del prodotto è stato ignorato lo 0); sono evidenziate le celle che contengono i rispettivi elementi neutri 0 e 1.

+	0	1	2	3	4	5	6	7	8
0	0	1	2	3	4	5	6	7	8
1	1	2	3	4	5	6	7	8	0
2	2	3	4	5	6	7	8	0	1
3	3	4	5	6	7	8	0	1	2
4	4	5	6	7	8	0	1	2	3
5	5	6	7	8	0	1	2	3	4
6	6	7	8	0	1	2	3	4	5
7	7	8	0	1	2	3	4	5	6
8	8	0	1	2	3	4	5	6	7

x	1	2	3	4	5	6	7	8
1	1	2	3	4	5	6	7	8
2	2	4	6	8	1	3	5	7
3	3	6	0	3	6	0	3	6
4	4	8	3	7	2	6	1	5
5	5	1	6	2	7	3	8	4
6	6	3	0	6	3	0	6	3
7	7	5	3	1	8	6	4	2
8	8	7	6	5	4	3	2	1

Nella tabella della somma si osserva che la distribuzione degli opposti è estremamente regolare ed è facile formulare una legge generale:

l'opposto di a in $\mathbf{Z}_n$ è $n-a$.

Nella tabella del prodotto si osserva innanzitutto che non tutti gli elementi hanno inverso (solo 2, 4, 5, 7 e 8 sono invertibili); come è noto, in $\mathbf{Z}_n$ ammettono inverso solo i numeri a primi con n, cioè tali che $\mathrm{MCD}(a, n) = 1$; il che significa che se (e solo se) n è primo allora tutti gli elementi (non nulli) di $\mathbf{Z}_n$ ammettono inverso. In altri termini $\mathbf{Z}_n$ è un **anello** per ogni n, ed è un **campo** se e solo se n è primo.

Inoltre la distribuzione degli inversi non mostra una regolarità evidente, a parte la simmetria rispetto alla diagonale principale (se $ab = 1$ allora $ba = 1$) e alla diagonale secondaria (se $ab = 1$ allora $\bar{a} \cdot \bar{b} = 1$).

Domanda: esiste una legge generale anche per l'inverso di a in $\mathbf{Z}_n$? Per esempio, qual è l'inverso di 1234 in $\mathbf{Z}_{1789}$? È possibile calcolarlo direttamente oppure dobbiamo calcolare

$$1234\cdot 1,\ 1234\cdot 2,\ 1234\cdot 3,\ \ldots$$

in $\mathbf{Z}_{1789}$ fino a che il risultato è 1?

La tabella del prodotto non fornisce indizi; proviamo allora con la tabella delle potenze, cioè eleviamo ogni elemento di $\mathbf{Z}_n$ (escluso 1, le cui potenze sono tutte uguali a 1) alle successive potenze 1, 2, …, $n-1$. Ecco la tabella delle potenze in $\mathbf{Z}_7$ e in $\mathbf{Z}_{11}$.

^	1	2	3	4	5	6
2	2	4	1	2	4	1
3	3	2	6	4	5	1
4	4	2	1	4	2	1
5	5	4	6	2	3	1
6	6	1	6	1	6	1

^	1	2	3	4	5	6	7	8	9	10
2	2	4	8	5	10	9	7	3	6	1
3	3	9	5	4	1	3	9	5	4	1
4	4	5	9	3	1	4	5	9	3	1
5	5	3	4	9	1	5	3	4	9	1
6	6	3	7	9	10	5	8	4	2	1
7	7	5	2	3	10	4	6	9	8	1
8	8	9	6	4	10	3	2	5	7	1
9	9	4	3	5	1	9	4	3	5	1
10	10	1	10	1	10	1	10	1	10	1

Si osservi l'ultima colonna: è composta di soli 1, cioè per ogni a non nullo risulta:

- in $\mathbf{Z}_7$: $a^6 = 1$;
- in $\mathbf{Z}_{11}$: $a^{10} = 1$.

Formuliamo la seguente congettura:

Per ogni $a \in \mathbf{Z}_n$, $a \neq 0$, risulta $a^{n-1} = 1$.

Se così fosse, avremmo trovato una legge generale per l'inverso di a in $\mathbf{Z}_n$: l'inverso di a sarebbe a^{n-2}, perché

$$a \cdot a^{n-2} = a^{n-1} = 1.$$

In realtà la congettura è corretta *se e solo se* ***n*** *è primo*. Questo è infatti quanto afferma il seguente, celebre teorema.

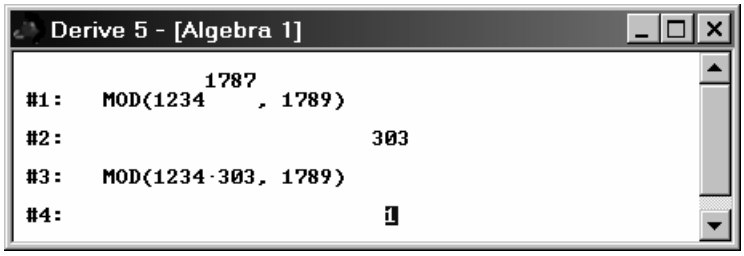
Teorema di Fermat

Se e solo se n è primo, per ogni $a \in \mathbf{Z}_n$ non nullo risulta $a^{n-1} = 1$.

Corollario.

Se n è primo, allora l'inverso di a ($a \neq 0$) in $\mathbf{Z}_n$ è a^{n-2}.

Dunque l'inverso di 1234 in $\mathbf{Z}_{1789}$ è $1234^{1787} = 303$.

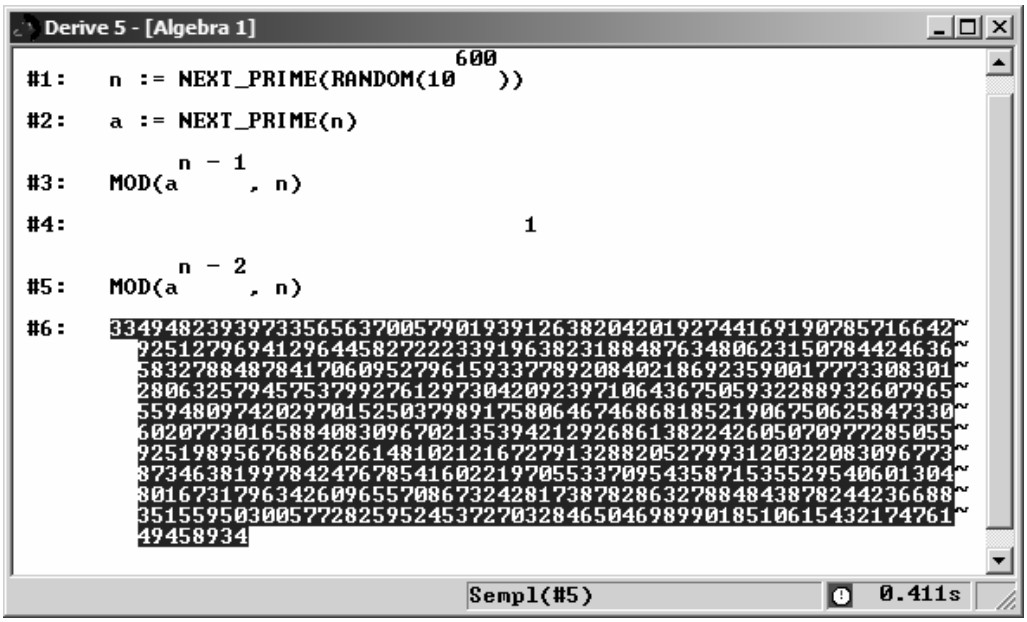


A questo punto è doverosa una domanda: se l'inverso di a in $\mathbf{Z}_n$ è a^{n-2}, calcolare l'inverso di a comporta necessariamente il prodotto di $n-2$ fattori uguali ad a? Se per calcolare l'inverso di a in $\mathbf{Z}_n$ dovessimo effettuare un numero di operazioni elementari circa uguale a n, i tempi di calcolo sarebbero proibitivi. Per esempio, se anche ogni prodotto venisse eseguito in 10^{-12} s, per elevare ad un esponente di 20 cifre (quindi "piccolo" per i calcoli che saranno richiesti da RSA), occorrerebbero $10^8 \text{ s} \approx 3$ anni; con un esponente di 30 cifre occorrerebbero 30 miliardi di anni.

Fortunatamente esistono algoritmi molto efficienti per calcolare la potenza di un numero, il cui tempo di calcolo è lineare rispetto alla lunghezza dell'esponente (vedi in Appendice A l'**algoritmo di Legendre**).

Nella figura seguente ci siamo procurati, con Derive, un numero primo $n \approx 10^{600}$, un numero $a < n$, e abbiamo calcolato in $\mathbf{Z}_n$:

- la potenza a^{n-1}, per verificare il teorema di Fermat;
- la potenza a^{n-2}, cioè l'inverso di a; come si vede in basso a destra, il tempo di calcolo è piccolo (circa 0.4 s).



Il Teorema di Eulero

Se il teorema di Fermat risolve il problema del calcolo dell'inverso di a in $\mathbf{Z}_n$ quando n è primo, rimane aperto il problema nel caso n non sia primo. Occorrerà aspettare più di un secolo perché questo problema venga risolto da Eulero.

Diamo ancora un'occhiata alla tabella delle potenze degli elementi di $\mathbf{Z}_n$, con n composto, per scoprire qualche eventuale regolarità. Le figure seguenti mostrano le potenze in $\mathbf{Z}_9$, $\mathbf{Z}_{14}$, $\mathbf{Z}_{15}$.

n = 9	^	1	2	3	4	5	6	7	8
	2	2	4	8	7	5	1	2	4
	3	3	0	0	0	0	0	0	0
	4	4	7	1	4	7	1	4	7
	5	5	7	8	4	2	1	5	7
	6	6	0	0	0	0	0	0	0
	7	7	4	1	7	4	1	7	4
	8	8	1	8	1	8	1	8	1

n = 14	^	1	2	3	4	5	6	7	8	9	10	11	12	13
	2	2	4	8	2	4	8	2	4	8	2	4	8	2
	3	3	9	13	11	5	1	3	9	13	11	5	1	3
	4	4	2	8	4	2	8	4	2	8	4	2	8	4
	5	5	11	13	9	3	1	5	11	13	9	3	1	5
	6	6	8	6	8	6	8	6	8	6	8	6	8	6
	7	7	7	7	7	7	7	7	7	7	7	7	7	7
	8	8	8	8	8	8	8	8	8	8	8	8	8	8
	9	9	11	1	9	11	1	9	11	1	9	11	1	9
	10	10	2	6	4	12	8	10	2	6	4	12	8	10
	11	11	9	1	11	9	1	11	9	1	11	9	1	11
	12	12	4	6	2	10	8	12	4	6	2	10	8	12
	13	13	1	13	1	13	1	13	1	13	1	13	1	13

n = 15	^	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	2	2	4	8	1	2	4	8	1	2	4	8	1	2	4
	3	3	9	12	6	3	9	12	6	3	9	12	6	3	9
	4	4	1	4	1	4	1	4	1	4	1	4	1	4	1
	5	5	10	5	10	5	10	5	10	5	10	5	10	5	10
	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6
	7	7	4	13	1	7	4	13	1	7	4	13	1	7	4
	8	8	4	2	1	8	4	2	1	8	4	2	1	8	4
	9	9	6	9	6	9	6	9	6	9	6	9	6	9	6
	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
	11	11	1	11	1	11	1	11	1	11	1	11	1	11	1
	12	12	9	3	6	12	9	3	6	12	9	3	6	12	9
	13	13	4	7	1	13	4	7	1	13	4	7	1	13	4
	14	14	1	14	1	14	1	14	1	14	1	14	1	14	1

Nella prima colonna sono evidenziati gli elementi di $\mathbf{Z}_n$ che ammettono inverso: si osservi che, per tutti gli elementi invertibili, risulta:

- in $\mathbf{Z}_9$: $a^6 = 1$
- in $\mathbf{Z}_{14}$: $a^6 = 1$ (e quindi $a^{12} = (a^6)^2 = 1$)
- in $\mathbf{Z}_{15}$: $a^4 = 1$ (e quindi $a^8 = (a^4)^2 = 1$ e $a^{12} = (a^4)^3 = 1$)

Sembra che anche in $\mathbf{Z}_n$, con n composto, esista un esponente (6 in $\mathbf{Z}_9$ e in $\mathbf{Z}_{14}$, 4 in $\mathbf{Z}_{15}$), tale che ogni elemento invertibile elevato a quell'esponente sia uguale a 1. Si tratta di capire qual è la relazione tra questo esponente e n. La risposta è nel seguente, altrettanto celebre, teorema, che generalizza il teorema di Fermat.

Teorema di Eulero

Per ogni $n \geq 2$ e per ogni $a \in \mathbf{Z}_n$ invertibile risulta

$$a^{\varphi(n)} = 1,$$

dove $\varphi(n)$ è il numero di naturali compresi tra 1 e n che sono primi con n.

Corollario

Per ogni $n \geq 2$ e per ogni $a \in \mathbf{Z}_n$ invertibile, cioè $\mathrm{MCD}(a, n) = 1$, l'inverso di a è

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

La funzione $\varphi(n)$ è chiamata **funzione di Eulero** ed associa ad ogni numero naturale n il numero di numeri $a \in \{1, 2, \ldots, n\}$ tali che $\mathrm{MCD}(a, n) = 1$. Per esempio, $\varphi(10) = 4$, poiché i numeri di $\mathbf{Z}_{10}$ primi con 10 sono 1, 3, 7, 9.

La tabella elenca i valori della funzione di Eulero per $n = 2, \ldots, 15$.

n	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\varphi(n)$	1	2	2	4	2	6	4	6	4	10	4	12	6	8

Se n è primo allora $\varphi(n) = n-1$, e dunque il teorema di Fermat è un caso particolare del teorema di Eulero.

Per la funzione φ è possibile dimostrare le seguenti proprietà.

- Se $\mathrm{MCD}(a, b) = 1$ allora $\varphi(ab) = \varphi(a)\varphi(b)$. Per esempio

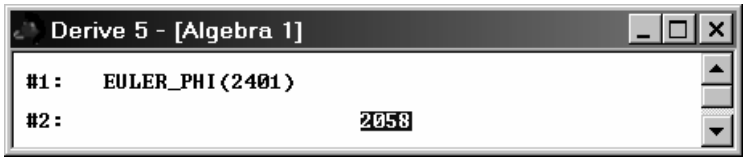
$$\varphi(72) = \varphi(8) \cdot \varphi(9) = 4 \cdot 6 = 24.$$

– Se p è primo allora $\varphi(p^m) = p^{m-1}(p-1)$. Per esempio,

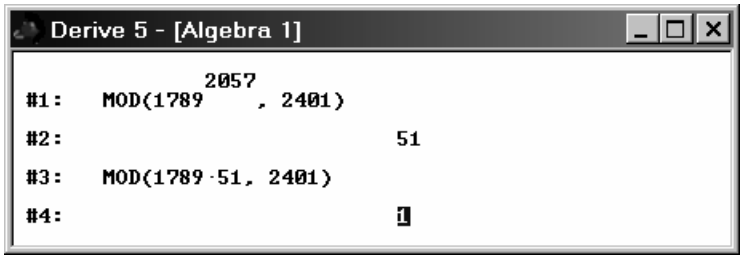
$$\varphi(2401) = \varphi(7^4) = 7^3 \cdot 6 = 343 \cdot 6 = 2058.$$

In particolare, se p e q sono due numeri primi distinti, allora $\varphi(pq) = (p-1) \cdot (q-1)$. Per esempio $\varphi(15) = \varphi(3)\varphi(5) = 2 \cdot 4 = 8$. Questa sarà una proprietà fondamentale che utilizzeremo nell'algoritmo RSA.

In Derive la funzione EULER_PHI(n) calcola $\varphi(n)$.



Noto il valore di $\varphi(n)$, possiamo calcolare l'inverso di qualsiasi elemento invertibile di $\mathbf{Z}_n$. Per esempio, poiché 1789 è primo, risulta MCD(1789, 2401) = = 1, e dunque 1789 è invertibile in $\mathbf{Z}_{2401}$. Poiché $\varphi(2401) = 2058$, l'inverso di 1789 è $1789^{2057} = 51$ in $\mathbf{Z}_{2401}$.



Mettiamo subito in evidenza che il calcolo di $\varphi(n)$ per numeri grandi non è realizzabile in tempi ragionevoli. Le proprietà della funzione φ mostrano che il calcolo di $\varphi(n)$ è immediato se si conosce la fattorizzazione di n; ma la fattorizzazione di n è esattamente ciò che verrà tenuto nascosto e che costituisce il motivo della inattaccabilità di RSA; le difficoltà di calcolare $\varphi(n)$ e di fattorizzare n sono equivalenti. Infatti:

– dalla fattorizzazione di n, $n = pq$ si ricava immediatamente $\varphi(n)$:

$$\varphi(n) = (p-1)(q-1);$$

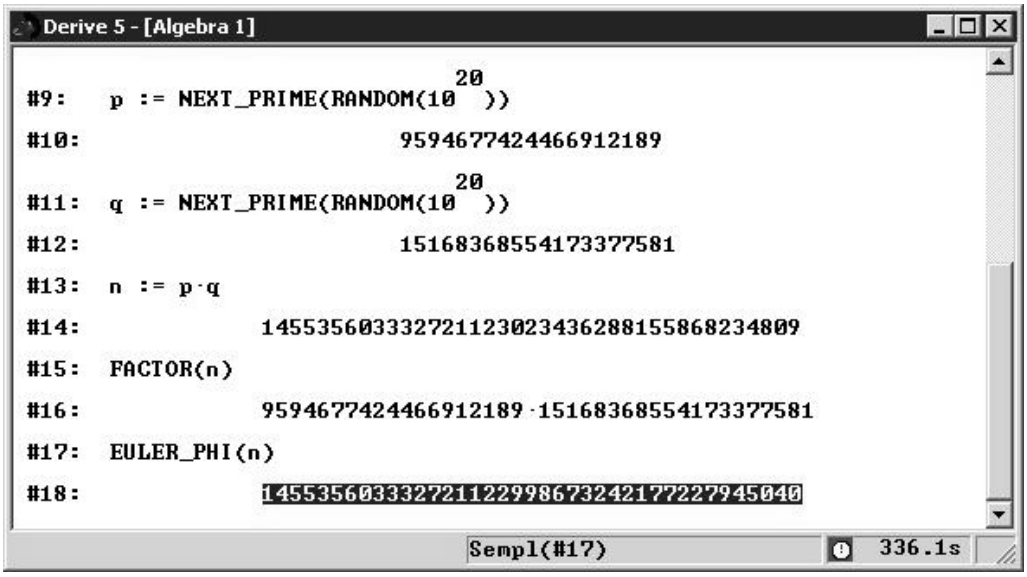
– dalla conoscenza di $\varphi(n)$ e n si ricava la fattorizzazione di n:

$$\varphi(n) = (p-1)(q-1) = pq - (p+q) + 1 = n - (p+q) + 1$$

da cui $p+q= n-\varphi(n)+1$; quindi di p e q si conosce il prodotto $pq=n$ e la somma $p+q=n-\varphi(n)+1$ ed è immediato calcolare p e q, che sono le soluzioni dell'equazione di secondo grado

$$x^2-(p+q)x+pq=0.$$

In effetti i tempi di calcolo di $\varphi(n)$ sono dello stesso ordine di grandezza della fattorizzazione di n. Per esempio con Derive abbiamo impiegato 336.4 secondi per fattorizzare il prodotto pq di due numeri primi casuali di 20 cifre, e 336.1 secondi per calcolare $\varphi(pq)$, come mostra la seguente figura.



L'algoritmo di Euclide per l'inverso di a in $\mathbf{Z}_n$

Fortunatamente non è necessario conoscere $\varphi(n)$ per calcolare l'inverso di a in $\mathbf{Z}_n$: un'opportuna applicazione dell'algoritmo di Euclide ci consente di effettuare il calcolo dell'inverso di a con strumenti del tutto differenti e in tempi rapidi.

Come è noto, l'algoritmo di Euclide è un algoritmo molto efficiente per il calcolo del MCD tra due numeri naturali a e b. Consiste nel calcolare la sequenza (strettamente decrescente)

$x_1 := a$
$x_2 := b$
$x_i := \mathrm{mod}(x_{i-1}, x_{i-2})$

fino a che, per un certo m, risulta $x_m = 0$. Allora x_{m-1} è il MCD tra a e b. L'algoritmo di Euclide, letto "al contrario", consente altresì di esprimere il numero MCD(a,b) come combinazione lineare di a e b, consente cioè di trovare due numeri interi $\boldsymbol{r}$ e $\boldsymbol{s}$ tali che

$$\boldsymbol{r}a+\boldsymbol{s}b = \mathrm{MCD}(a, b).$$

Vediamo un esempio. Calcoliamo MCD(1789, 1234) utilizzando l'algoritmo di Euclide e tenendo conto dei successivi quozienti q_i.

$\boldsymbol{x_i}$	1789	1234	555	124	59	6	5	1	0
$\boldsymbol{q_i}$		1	2	4	2	9	1	5	

Innanzitutto risulta $x_8 = \mathrm{MCD}(1789, 1234) = 1$.

A partire dal MCD possiamo esprimere ciascun x_j come combinazione lineare di x_{i-2} e x_{i-1} (in grassetto i coefficienti della combinazione lineare):

$1 = 6–\mathbf{1}\cdot5$
$5 = 59–\mathbf{9}\cdot6$
$6 = 124–\mathbf{2}\cdot59$
$59 = 555–\mathbf{4}\cdot124$
$124 = 1234–\mathbf{2}\cdot555$
$555 = 1789–\mathbf{1}\cdot1234$

Ora, sostituendo nella prima uguaglianza il risultato della seconda, della terza, e così via, otteniamo:

$1 = 6–\mathbf{1}\cdot5$
$= 6–\mathbf{1}\cdot(59–\mathbf{9}\cdot6) = –\mathbf{1}\cdot59+\mathbf{10}\cdot6$
$= –\mathbf{1}\cdot59+\mathbf{10}\cdot(124–\mathbf{2}\cdot59) = \mathbf{10}\cdot124–\mathbf{21}\cdot59$
$= \mathbf{10}\cdot124–\mathbf{21}\cdot(555–\mathbf{4}\cdot124) = –\mathbf{21}\cdot555+\mathbf{94}\cdot124$
$= –\mathbf{21}\cdot555+\mathbf{94}\cdot(1234–\mathbf{2}\cdot555) = \mathbf{94}\cdot1234–\mathbf{209}\cdot555$
$= \mathbf{94}\cdot1234–\mathbf{209}\cdot(1789–\mathbf{1}\cdot1234) = –\mathbf{209}\cdot1789+\mathbf{303}\cdot1234$

Dunque

$$-\mathbf{209}\cdot 1789+\mathbf{303}\cdot 1234 = 1.$$

Ora leggiamo questa combinazione lineare in $\mathbf{Z}_{1789}$:

$$\mathbf{303}\cdot 1234 = 1-\mathbf{209}\cdot 1789$$

$$\mathbf{303}\cdot 1234 = 1 \mod 1789$$

il che è come dire che 303 è l'inverso di 1234 in $\mathbf{Z}_{1789}$.

Riassumendo: in $\mathbf{Z}_n$ (sia che n sia primo o composto), a è invertibile se e solo se MCD(n, a) = 1. Utilizzando l'algoritmo di Euclide possiamo esprimere 1 come combinazione lineare di n e a:

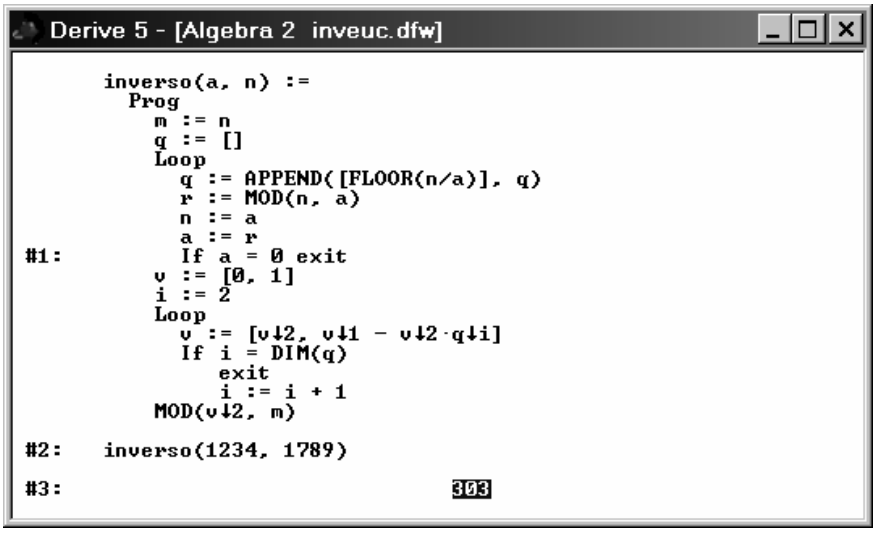
$$\boldsymbol{r}n+\boldsymbol{s}a = 1$$

da cui

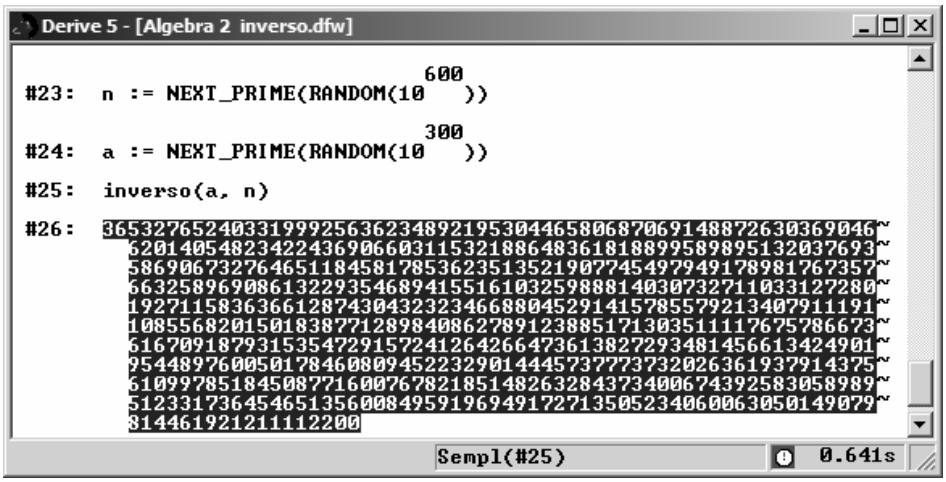
$$\boldsymbol{s}a = 1 \mod n,$$

cioè s è l'inverso di a in $\mathbf{Z}_n$.

Con la faticosa sintassi di programmazione di Derive (per programmare è molto meglio usare la TI-89), abbiamo costruito la funzione `INVERSO(a,n)`. Nella funzione non viene eseguito un controllo sul fatto che risulti MCD(a, n) = 1: si dà per ipotesi che a e n siano primi tra loro, in modo che a sia effettivamente invertibile in $\mathbf{Z}_n$.



Con questo algoritmo i tempi di calcolo sono molto ridotti. La figura seguente mostra il calcolo dell'inverso di $a \approx 10^{300}$ in $\mathbf{Z}_n$ con $n \approx 10^{600}$ in circa 0.6 s.



In effetti si può dimostrare che il numero di operazioni elementari di questo algoritmo è lineare rispetto alla lunghezza di n.

In Appendice B è descritta in maggior dettaglio l'implementazione dell'algoritmo.

L'algoritmo RSA: dalla parte della Certification Authority

Abbiamo tutto quello che ci serve per descrivere l'algoritmo di crittazione RSA. Articoliamo la descrizione in due fasi: la scelta delle chiavi (svolta dalla CA) e l'algoritmo di codifica-decodifica, svolto dagli utenti.

La scelta delle chiavi (pubblica e privata) da assegnare all'utente A da parte della Certification Authority si fonda sulla possibilità di procurarsi due numeri primi p e q "grandi"; oggi l'ordine di grandezza standard di p e q è circa 10^{300}. Come abbiamo visto, questo non è un problema dal punto di vista computazionale.

Dunque la CA:

- si procura due numeri primi p e q;
- calcola il loro prodotto $n = pq$;
- calcola $\varphi(n) = (p-1)(q-1)$; il numero n viene reso pubblico, ma non la sua fattorizzazione pq, né $\varphi(n)$.

Per ciascun utente si costruiscono ora due chiavi, quella pubblica H e quella privata K. Una chiave, per esempio quella pubblica, è un qualunque numero H minore di $\varphi(n)$ e primo con $\varphi(n)$ (e quindi invertibile in $\mathbf{Z}_{\varphi(n)}$). L'altra chiave, quella privata, è l'inverso di H in $\mathbf{Z}_{\varphi(n)}$.

Svolgiamo un esempio con numeri "piccoli": partiamo dai numeri primi $p := 1789$ e $q := 1801$; risulta

$$n = 3\,221\,989 \qquad \text{e} \qquad \varphi(n) = 3\,218\,400.$$

Prendiamo come H un qualsiasi numero primo con $\varphi(n)$, per esempio il numero primo

$$H := 1\,093\,531$$

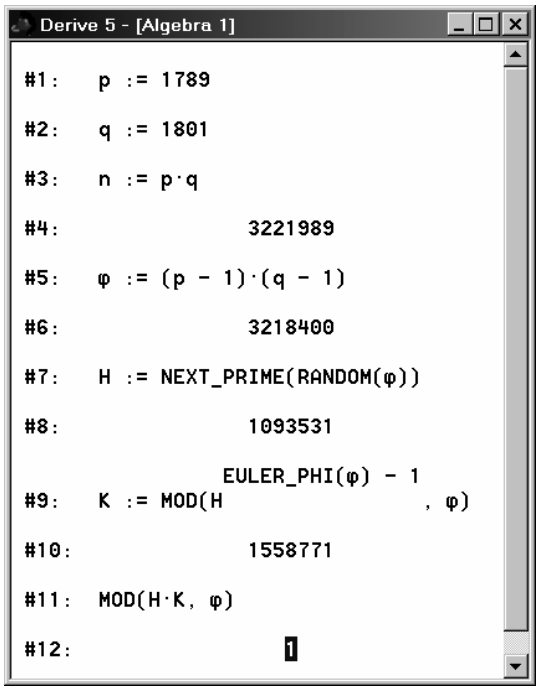
Ora calcoliamo $K = H^{-1}$ in $\mathbf{Z}_{3218400}$. Poiché $\varphi(n)$ è "piccolo", si fattorizza rapidamente, e possiamo perciò applicare direttamente il teorema di Eulero: per calcolare K dobbiamo elevare H a $\varphi(3\,218\,400)-1$ in $\mathbf{Z}_{\varphi(n)}$. Utilizzando Derive risulta

$$\varphi(\varphi(n))-1 = \varphi(3\,218\,400)-1 = 852\,479$$

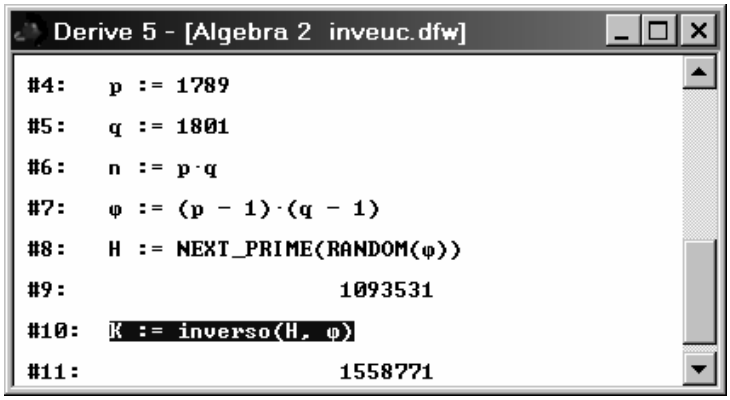
e dunque

$$K = H^{852\,479} \pmod{3\,218\,400} = 1\,558\,771.$$

Ecco i calcoli con Derive.



Con l'algoritmo di Euclide arriviamo, per K, allo stesso risultato, come mostra l'applicazione della funzione INVERSO che abbiamo costruito in Derive.



La prima fase è terminata. La CA consegna all'utente A:

- la chiave H = 1 093 531, che viene resa pubblica insieme al valore $n = 3\,221\,989$;
- la chiave privata K = 1 558 771, che A tiene nascosta.

Un estraneo conosce dunque H e n, ma non conosce K, né $\varphi(n)$. Se vuole conoscere K (e questo, come vedremo, è l'unico modo di decodificare il messaggio) deve fattorizzare n, impresa impossibile in tempi ragionevoli se p e q sono numeri primi "grandi".

L'algoritmo RSA: dalla parte dell'utente

La proprietà fondamentale che lega H, K, n (e che costituisce il cuore dell'algoritmo RSA) è la seguente: per ogni $x \in \mathbf{Z}_n$ risulta

$$\left(x^{\mathrm{H}}\right)^{K} = x \qquad \mathrm{mod}\ n.$$

In altri termini, se MCD(H, $\varphi(n)$) = 1, cioè se H è invertibile in $\mathbf{Z}_{\varphi(n)}$ allora la funzione

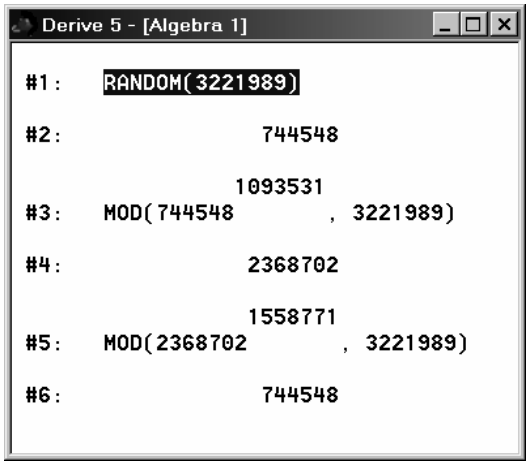
$$f(x) := x^{\mathrm{H}}$$

è invertibile in $\mathbf{Z}_n$, e la sua funzione inversa è

$$f^{-1}(x) = x^{\mathrm{K}},$$

dove HK = 1 in $\mathbf{Z}_{\varphi(n)}$.

Illustriamo la proprietà con i valori H, K, n dell'esempio precedente; scegliamo a caso con Derive un numero $x \in \mathbf{Z}_n$, calcoliamo x^{H}, $\left(x^{\mathrm{H}}\right)^{K}$ e verifichiamo che otteniamo di nuovo x.



La proprietà è facile da dimostrare se x è primo con n; infatti se $HK = 1$ in $\mathbf{Z}_{\varphi(n)}$ allora in $\mathbf{Z}_n$ risulta

$$\left(x^{\mathrm{H}}\right)^{K} = x^{\mathrm{HK}} = x^{t\varphi(n)+1} = x^{t\varphi(n)}\,x = \left(x^{\varphi(n)}\right)^{t} x\,;$$

ma per il teorema di Eulero $x^{\varphi(n)} = 1$, dunque

$$\left(x^{\varphi(n)}\right)^{t} x = 1^{t}\,x = x.$$

Se x non è primo con n la dimostrazione è solo un po' più laboriosa.

Bene, abbiamo trovato le due funzioni f (facile da calcolare) e f^{-1} (praticamente impossibile da calcolare) che ci servono per l'algoritmo. La funzione f è la potenza con esponente H in $\mathbf{Z}_n$, la funzione inversa f^{-1} è la potenza con esponente K (l'inverso di H in $\mathbf{Z}_{\varphi(n)}$) in $\mathbf{Z}_n$. H è noto a tutti e quindi chiunque può mandare un messaggio segreto ad A, ma solo A può decodificarlo perché solo lui conosce la chiave segreta K.

Riassumendo, le quattro funzioni g, g^{-1}, f, f^{-1} per l'algoritmo RSA sono le seguenti:

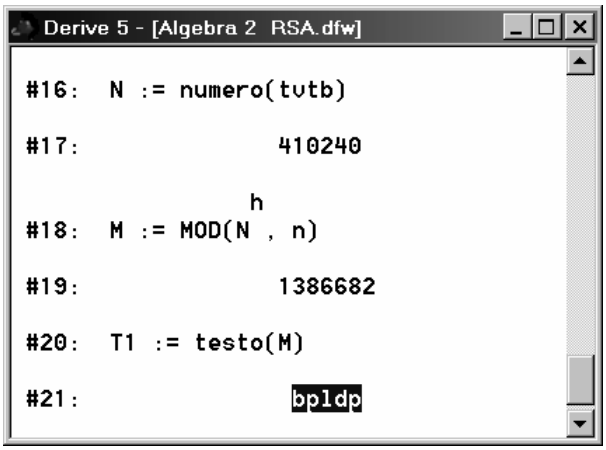
- g(T) è la funzione `numero`;
- g^{-1}(N) è la funzione `testo`;
- f(N) è la funzione x^{H} in $\mathbf{Z}_n$, dove H è la chiave pubblica dell'utente;
- f^{-1}(M) è la funzione x^{K} in $\mathbf{Z}_n$, dove K è l'inverso di H in $\mathbf{Z}_{(p-1)(q-1)}$ ed è la chiave privata dell'utente.

Vediamo l'algoritmo al lavoro utilizzando le chiavi H e K precedenti. Supponiamo che B voglia mandare il messaggio segreto

T := "tvtb"

ad A. B legge sull'elenco pubblico la chiave pubblica H di A e il valore n. Poi calcola

$$g^{-1}(f(g(\mathrm{T}))).$$



Risulta

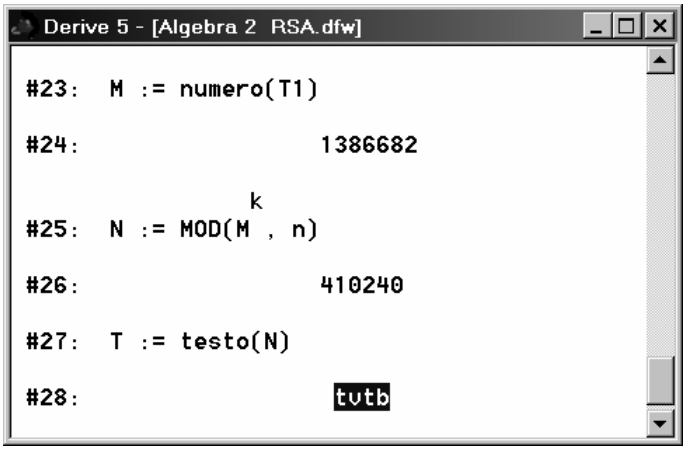
$N = g(T) =$ `numero("tvtb")` $= 410\,240$.

$M = f(N) = N^H \bmod n = 410\,240^{1\,093\,531} \bmod 3\,221\,989 = 1\,386\,682$.

$T' = g^{-1}(M) =$ `testo(M)` $=$ `testo(1386682)` $=$ `"bpldp"`.

B manda dunque ad A il testo codificato T' = "bpldp".
A riceve T' e calcola

$$g^{-1}(f^{-1}(g(T'))).$$



Risulta

$M =$ `numero("bpldp")` $= 1\,386\,682$

$$N = f^{-1}(M) = M^K \bmod n = 1\,386\,682^{1\,558\,771} \bmod n = 410\,240$$

$$T = \texttt{testo(N)} = \texttt{testo(410240)} = \text{“tvtb”}$$

e finalmente A legge il messaggio decodificato “tvtb”.

Ovviamente, con p e q così piccoli, la segretezza è facilmente scardinabile: si fattorizza n (Derive impiega 0.016 secondi) e si ricava $p = 1789$ e $q = 1801$ e di conseguenza $\varphi(n) = 1788 \cdot 1800 = 3\,218\,400$.

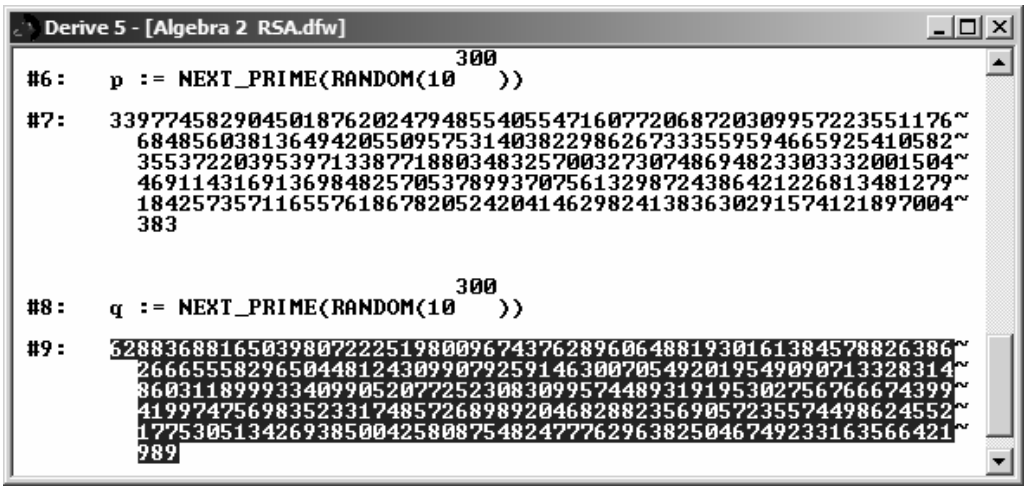
Poiché H è noto, è immediato calcolare $K = H^{-1}$ in $\mathbf{Z}_{\varphi(n)} = \mathbf{Z}_{3218400}$.

Formuliamo invece un esempio con numeri primi talmente grandi che la decodifica sia “impossibile”.

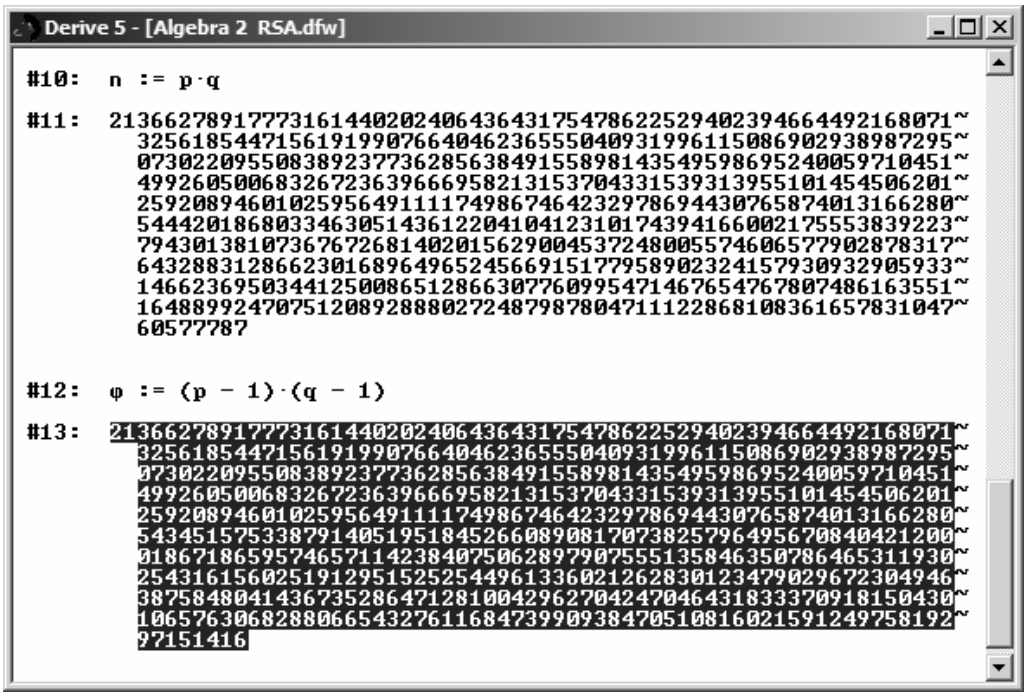
Un esempio con numeri “grandi”

L’esempio che ora svolgiamo parte da due numeri primi che hanno l’ordine di grandezza pari a quello utilizzato oggi nella pratica. Vogliamo mostrare che Derive è più che sufficiente per i calcoli necessari allo scopo.

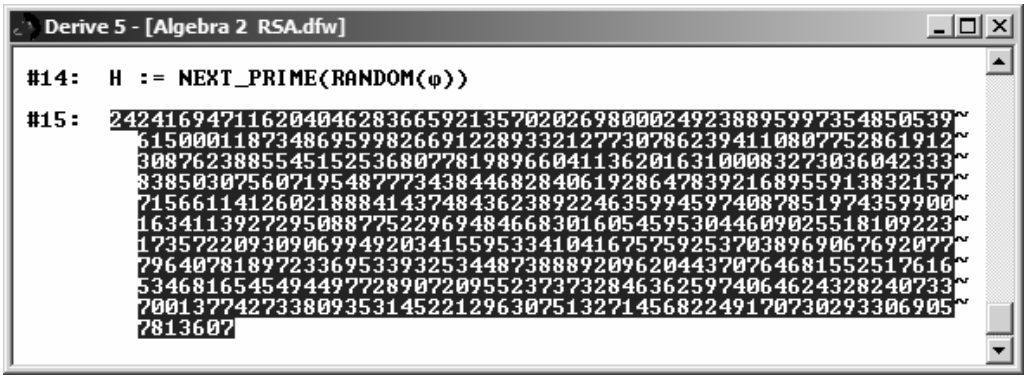
Svolgiamo innanzitutto il lavoro della CA. Da quanto abbiamo visto all’inizio la fattorizzazione del prodotto di due numeri primi di 300 cifre è “impossibile”. Procuriamoci dunque due numeri primi casuali p e q dell’ordine di grandezza di 10^{300}. Il comando NEXT_PRIME di Derive se la cava egregiamente: il tempo d’attesa è praticamente nullo.



Ora calcoliamo n e $\varphi(n)$, il cui ordine di grandezza è 10^{600}.

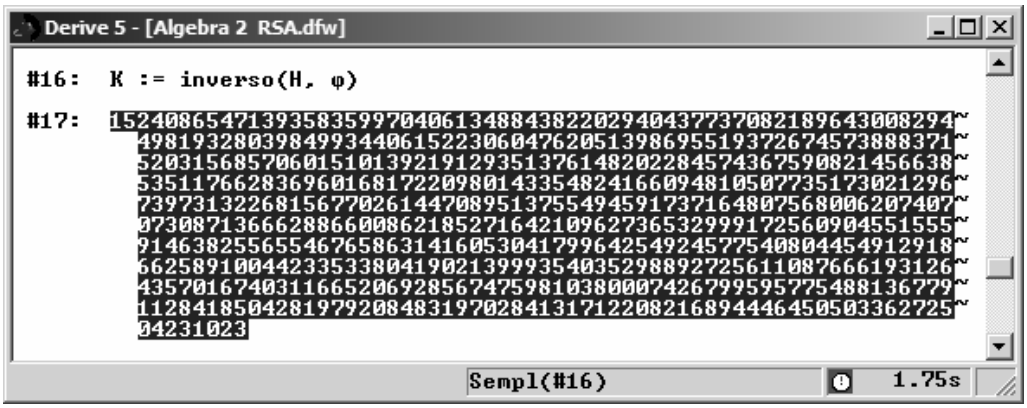


Ci procuriamo, come chiave pubblica H, un numero casuale qualsiasi primo con $\varphi(n)$; per essere certi che H sia primo con $\varphi(n)$, utilizziamo ancora NEXT_PRIME, scegliendo casualmente un numero primo minore di $\varphi(n)$.



Veniamo ora al compito primario della CA, cioè calcolare la chiave privata K che, ricordiamo, è l'inverso di H in $\mathbf{Z}_{\varphi(n)}$. Sarebbe impensabile calcolare K con il teorema di Eulero, dato che $\varphi(n) \approx 10^{600}$, è troppo grande. Utilizziamo dunque la funzione da noi costruita INVERSO(H, φ) (vedi Appendice B).

Con Derive K viene calcolato in meno di 2 secondi (si guardi in basso a destra della figura seguente).



La Certification Autority manda all'utente A, che ne ha fatto richiesta, la chiave pubblica H, la chiave privata K e il numero *n*; H e *n* vengono resi pubblici.

B vuole mandare ad A il seguente messaggio:

```
T := "dai diamanti non nasce niente
     dal letame nascono i fior"
```

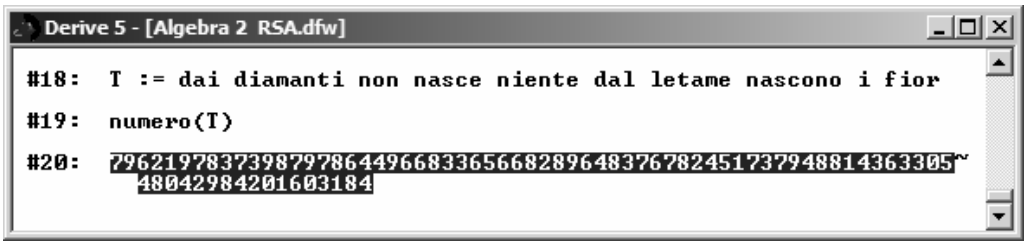
Innanzitutto occorre assicurarsi che risulti

$$\text{numero(T)} < n.$$

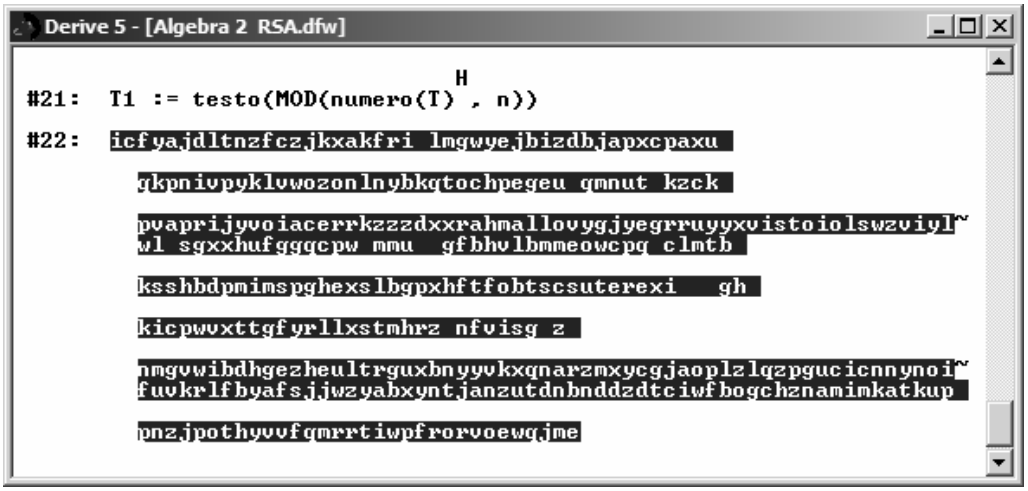
Infatti, poiché tutte le operazioni vengono svolte in $\mathbf{Z}_n$, per garantire la biunivocità della corrispondenza testo-numero occorre che il numero corrispondente ad un messaggio sia minore di *n*; altrimenti, poco male: è sufficiente suddividere il messaggio in più sottomessaggi. A questo scopo, con $n \approx 10^{600}$ e con un alfabeto di 27 caratteri, è sufficiente che il messaggio abbia un numero di caratteri (spazi compresi) minore di

$$\log_{27}(10^{600}) \approx 419 \text{ caratteri.}$$

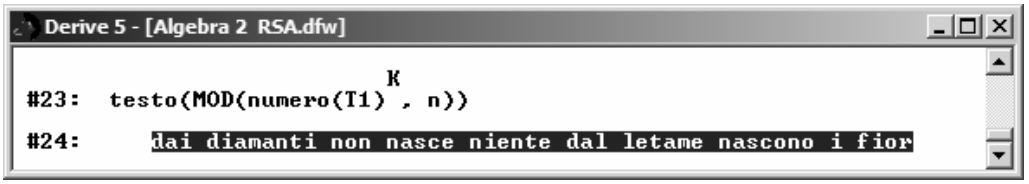
Nel nostro esempio il numero del messaggio è enormemente minore di *n*, quindi la biunivocità è assicurata. Trasformiamo T in numero.



Ora la codifica: B eleva numero(T) alla H modulo *n*, e ritrasforma il risultato in testo. La figura seguente mostra il testo che riceve A.



A riceve il testo T1, lo trasforma in numero, eleva il risultato alla K (che solo lui conosce) modulo *n*, e ritrasforma il risultato in testo.



Un malintenzionato C che volesse decifrare il testo dovrebbe conoscere K, cioè l'inverso di H in $\mathbf{Z}_{\varphi(n_A)}$; conosce H, ma non conosce φ(*n*) e l'unico modo per ottenere φ(*n*) è fattorizzare *n*. Ma *n* ha 600 cifre, è il prodotto di due primi di 300 cifre, la sua fattorizzazione è impossibile: la segretezza è assicurata.

Segretezza e firma digitale

Il meccanismo che abbiamo appena illustrato permette di ottenere la **segretezza** di un messaggio: nessun malintenzionato può decifrare il messaggio mandato da B ad A. Ma questo non è l'unico scopo della crittografia. Per esempio, quando A riceve il messaggio è sicuro che nessun altro tranne lui può averlo letto, ma non può essere sicuro che sia stato proprio B a mandarlo. Di norma il messaggio codificato è accompagnato da un messaggio in chiaro che ne illustra lo scopo e dichiara il mittente. Un malintenzionato C potrebbe scrivere alla banca A, dichiarando di essere B, e di voler bonificare una somma di denaro a C. In questo caso ciò che importa non è tanto la segretezza del messaggio (l'intenzione di B bonificare una certa somma a C) quanto l'autenticità del mittente.

I sistemi a chiave pubblica consentono di risolvere questo problema: un utente può apporre, al messaggio codificato che manda, la cosiddetta **firma digitale**. Se B vuole mandare ad A un messaggio di cui non gli importa tanto la segretezza (chiunque lo può leggere), quanto il fatto che A sia certo che solo B può averglielo mandato, basta modificare l'ordine di codifica; vediamo come.

Anche B chiede alla Certification Authority una chiave pubblica H_B e una chiave privata K_B e il corrispondente numero n_B. Codifica il testo T *con la propria chiave privata* K_B e lo manda ad A. A legge nel messaggio in chiaro chi è il mittente. Si procura la chiave pubblica H_B di B e decodifica il testo. A è sicuro che solo B può avergli mandato quel testo, poiché è l'unico che possiede la chiave privata K_B. Naturalmente il testo mandato da B ad A può essere decodificato da chiunque (poiché serve solo la chiave pubblica di B), ma lo scopo non era in questo caso la **segretezza** ma la **firma digitale** da parte di B.

È possibile ottenere sia la segretezza sia la firma digitale? Sì, la simmetria delle funzioni f e f^{-1} consente a B di mandare un messaggio ad A con le seguenti caratteristiche:

- solo A può leggerlo: il messaggio è segreto;
- solo B può averlo mandato: il messaggio possiede la firma digitale.

Funziona in questo modo: B codifica il messaggio prima *con la propria chiave privata*, e poi codifica il risultato *con la chiave pubblica di A*. A riceve il messaggio, lo decodifica con la propria chiave privata (solo A può farlo, il messaggio è segreto) e poi decodifica il risultato con la chiave pubblica di B (solo B può averlo mandato, il messaggio ha la firma digitale).

In questo modo la comunicazione tra A e B è perfettamente tutelata da occhi indiscreti ed è sempre garantita l'identità sia del mittente sia del destinatario.

Conclusioni

Il funzionamento dell'algoritmo RSA è sancito, come abbiamo visto, da una rete di proprietà aritmetiche e costituisce dunque un notevole **successo** della matematica, proprio in un settore, l'aritmetica, tradizionalmente lontano dalle applicazioni. A questo proposito, solo nel 1940 Godfrey Hardy, nel suo *Apologia di un matematico*, scriveva:

"*È giusto che Gauss e anche gli altri matematici minori si rallegrino per almeno una scienza, e proprio la loro [la Teoria dei Numeri], che il distacco stesso dalle contingenze umane conserva benigna e pulita*"

Anche per questo motivo l'algoritmo RSA rappresenta un grande successo dell'aritmetica: è paradossale, tuttavia, che la segretezza che l'algoritmo RSA promette (e mantiene) sia garantita proprio da una **sconfitta** della matematica: il non saper fattorizzare in tempi ragionevoli il prodotto di due numeri primi grandi.

APPENDICE A

L'algoritmo di Legendre

Come abbiamo visto, l'algoritmo RSA prevede che si sappia calcolare rapidamente una potenza a^b in $\mathbf{Z}_n$, con b ed n numeri grandi. Esiste un algoritmo, noto come *algoritmo di Legendre*, che consente di effettuare un numero di operazioni elementari che è lineare rispetto al numero di cifre di b, cioè è lineare rispetto alla lunghezza dell'input.

Per capirne il funzionamento, supponiamo per semplicità che b sia una potenza di 2, $b = 2^c$. Per calcolare a^b sarebbero necessari allora non b prodotti, ma c elevamenti al quadrato, cioè c prodotti:

$$a^b = a^{\left(2^c\right)} = a^{2\cdot 2\cdot \ldots \cdot 2}\left(\left(\left(a^2\right)^2\right)^{\cdots}\right)^2$$

e quindi il numero di operazioni elementari sarebbe $c = \log_2(b)$. Con

$$b \approx 10^{600} \approx 2^{2000},$$

dovremmo calcolare soltanto 2000 operazioni elementari; se anche ciascuna richiedesse ben 10^{-6} s, il tempo di calcolo sarebbe due millesimi di secondo.

Se b non è una potenza di 2, come accade in generale, è sufficiente un piccolo aggiustamento; vediamo ad esempio come calcolare a^{100}. Il numero 100 in base 2 si scrive

1100100;

si pone inizialmente la variabile z (che conterrà il risultato) uguale a 1 e la variabile x uguale ad a. Si scorrono le cifre di 1100100 a partire dall'ultima a destra: se la cifra è 0 si eleva x al quadrato e si mette il risultato in x; se la cifra è 1 si moltiplica z per x e si mette il risultato in z.

Se poi il calcolo avviene in $\mathbf{Z}_n$, avremo premura di ridurre il risultato modulo n ad ogni moltiplicazione.

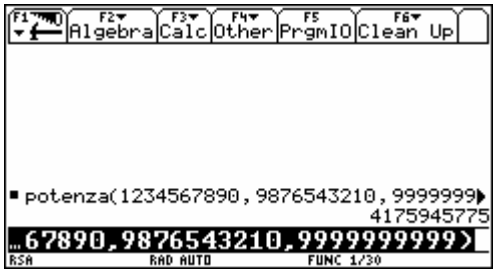
Nel linguaggio TI-Basic l'algoritmo di Legendre è così implementato:

```
potenza(x,y,n)
Func
Local  z
1→z
While  y>0
If mod(y,2)=1 Then
mod(z*x,n)→z
y−1→y
EndIf
y/2→y
mod(x*x,n)→x
EndWhile
z
EndFunc
```

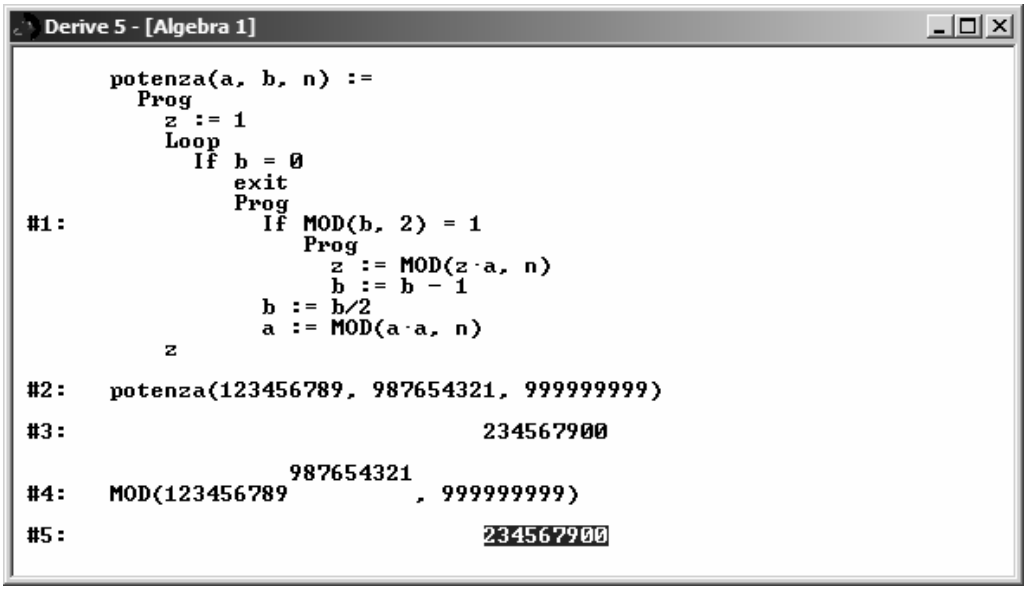
La figura seguente mostra il calcolo di

$$1234567890^{9876543210} \bmod 9999999999,$$

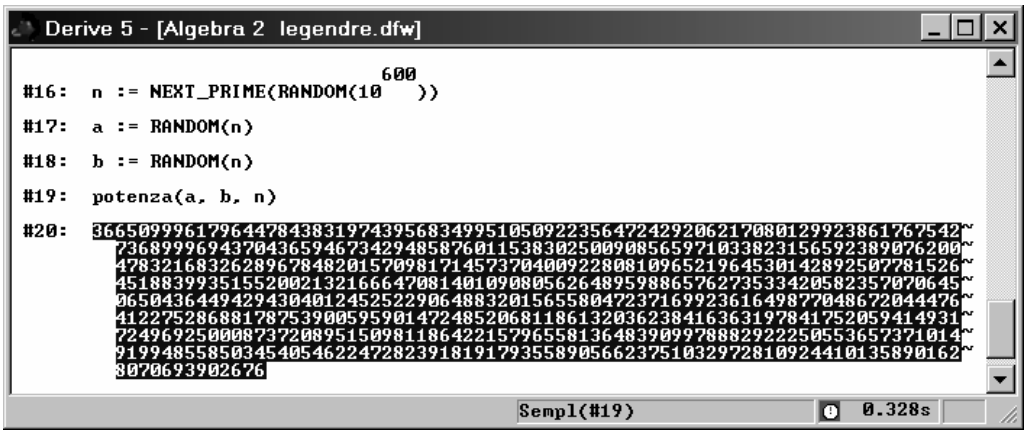
che viene effettuato dalla TI-89 in circa 2.5 secondi.



Con Derive, invece, la programmazione è come al solito faticosa; l'algoritmo è illustrato nella seguente figura.



Il tempo richiesto per il calcolo di $a^b \bmod n$, con n di 600 cifre, è circa 3 decimi di secondo.



APPENDICE B

L'inverso di a in $\mathbf{Z}_n$ con l'algoritmo euclideo

Abbiamo già visto come implementare il calcolo dell'inverso di a in $\mathbf{Z}_n$ (se MCD$(a, n) = 1$) con Derive. Vediamone i dettagli con il linguaggio di programmazione della TI-89.

Riprendiamo l'esempio del calcolo dell'inverso di 1234 in $\mathbf{Z}_{1789}$. Ci procuriamo innanzitutto la sequenza dei resti $x_1, \ldots, x_8$ e dei quozienti $q_1, \ldots, q_6$:

x_i	1789	1234	555	124	59	6	5	1
q_i		1	2	4	2	9	1	

Ora analizziamo il modo in cui $x_8 = 1$ si esprime come combinazione lineare di x_6 e x_7, di x_5 e x_6, e così via fino ad esprimere 1 come combinazione lineare di $x_1 = 1789$ e $x_2 = 1234$ (nella colonna a destra abbiamo messo in evidenza le coppie dei coefficienti):

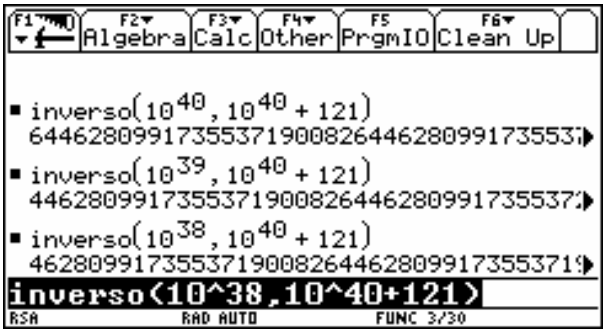
$1 = \mathbf{0}\cdot 5 + \mathbf{1}\cdot 1$	[0, 1]
$= \mathbf{1}\cdot 6 - \mathbf{1}\cdot 5$	[1, ⁻1]
$= {}^{-}\mathbf{1}\cdot 59 + \mathbf{10}\cdot 6$	[⁻1, 10]
$= \mathbf{10}\cdot 124 - \mathbf{21}\cdot 59$	[10, ⁻21]
$= {}^{-}\mathbf{21}\cdot 555 + \mathbf{94}\cdot 124$	[⁻21, 94]
$= \mathbf{94}\cdot 1234 - \mathbf{209}\cdot 555$	[94, ⁻209]
$= {}^{-}\mathbf{209}\cdot 1789 + \mathbf{303}\cdot 1234$	[⁻209, 303]

Scopriamo che la legge di formazione dei coefficienti è la seguente: si parte dalla coppia [0,1] e se $[r, s]$ sono i coefficienti della combinazione lineare di 1 in funzione di x_{i-1} e x_i allora $[s, r-s\cdot q_i]$ sono i coefficienti della combinazione lineare di 1 in funzione di x_i e x_{i+1}.

Ecco l'implementazione dell'algoritmo nel linguaggio della TI-89.

```
inverso(a,n)
Func
Local r,c,q,i,m
n→m:{}→q
Loop
augment({floor(m/a)},q)→q
mod(m,a)→r
a→m:r→a
If a=0 Then
Exit
EndIf
EndLoop
{0,1}→c
For i,2,dim(q)
{c[2],c[1]-c[2]*q[i]}→c
EndFor
mod(c[2],n)
EndFunc
```

I tempi di calcolo con la TI-89 sono molto contenuti; per calcolare l'inverso di a in $\mathbf{Z}_n$, con n di 40 cifre, sono sufficienti un paio di secondi.



APPENDICE C

L'algoritmo di Solovay-Strassen per la primalità

Il teorema di Fermat afferma che se n è primo (e solo in questo caso) allora per ogni $a = 1, 2, \ldots, n-1$ risulta

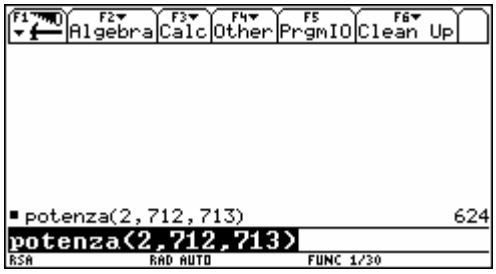
$$a^{n-1} = 1 \text{ in } \mathbf{Z}_n.$$

Dunque, se per un certo $a \in \mathbf{Z}_n$ non nullo risulta $a^{n-1} \neq 1$ allora n è certamente composto: il numero a è così un **testimone contro** la primalità di n. Per esempio:

713 è primo?

Prendiamo il testimone più piccolo, cioè $a = 2$; risulta in $\mathbf{Z}_{713}$:

$$2^{712} = 624 \neq 1.$$



Quindi 713 è composto (= 23·31). Si osservi che in questo modo possiamo dichiarare 713 composto *senza conoscerne alcun fattore*.

E se invece, per un certo a, risulta $a^{n-1} = 1$, possiamo affermare che n è primo? La risposta è no, e il più piccolo controesempio è $n = 341$. Infatti

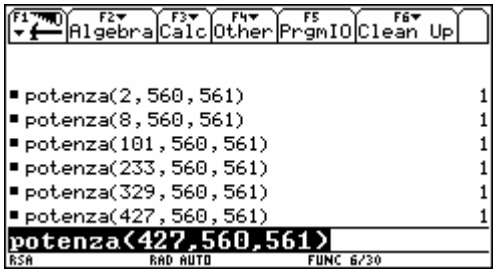
$$2^{340} = 1 \text{ in } \mathbf{Z}_{341}$$

ma 341 è composto (= 11·31). Insomma, il numero 2 è un **testimone a favore** della primalità di 341, ma è un testimone che si rivela inattendibile.

E se aumentiamo il numero di testimoni a favore? Può essere che, pur di aumentare il numero di testimoni, si possa dichiarare n primo, almeno con una certa probabilità?

Anche in questo caso la risposta è no. Infatti esistono infiniti numeri compositi, chiamati **numeri di Carmichael**, per i quali *tutti* gli a primi con n so-

no testimoni a favore della primalità e sono tutti inattendibili. Il più piccolo numero di Carmichael è $561 = 3 \cdot 11 \cdot 17$ (i successivi sono 1105, 1729, 2465, 2821, 6601, 8911, …). È ovvio che se si prende un testimone a che non sia primo con 561 non può risultare $a^{560} = 1$ in $\mathbf{Z}_{561}$ (altrimenti a sarebbe invertibile). Sappiamo che i numeri primi con 561 sono $\varphi(561) = 320$: bene, per *tutti* questi risulta $a^{560} = 1$. Ecco alcuni esempi.



Quindi il teorema di Fermat può essere utilizzato per affermare che un certo n è composto, ma non per riconoscere se n è primo.

Cerchiamo qualche altra proprietà. Si osservino le seguenti tabelle delle potenze in $\mathbf{Z}_7$ e $\mathbf{Z}_{11}$.

n = 7	^	1	2	3	4	5	6
	1	1	1	1	1	1	1
	2	2	4	1	2	4	1
	3	3	2	6	4	5	1
	4	4	2	1	4	2	1
	5	5	4	6	2	3	1
	6	6	1	6	1	6	1

n = 11	^	1	2	3	4	5	6	7	8	9	10
	1	1	1	1	1	1	1	1	1	1	1
	2	2	4	8	5	10	9	7	3	6	1
	3	3	9	5	4	1	3	9	5	4	1
	4	4	5	9	3	1	4	5	9	3	1
	5	5	3	4	9	1	5	3	4	9	1
	6	6	3	7	9	10	5	8	4	2	1
	7	7	5	2	3	10	4	6	9	8	1
	8	8	9	6	4	10	3	2	5	7	1
	9	9	4	3	5	1	9	4	3	5	1
	10	10	1	10	1	10	1	10	1	10	1

Si nota che:

- in $\mathbf{Z}_7$ le potenze terze sono tutte uguali a 1 oppure a 6 (che è $^{-}1$ in $\mathbf{Z}_7$);
- in $\mathbf{Z}_{11}$ le potenze quinte sono tutte uguali a 1 oppure a 10 (cioè $^{-}1$ in $\mathbf{Z}_{11}$).

Si tratta di una proprietà generale: se n è primo allora per ogni $a \in \mathbf{Z}_n$ non nullo risulta

$$a^{\frac{n-1}{2}} = 1 \qquad \text{oppure} \qquad a^{\frac{n-1}{2}} = {}^{-}1.$$

Infatti $a^{n-1} = \left(a^{\frac{n-1}{2}} \right)^2$; se n è primo allora per il teorema di Fermat

$$a^{n-1} = \left(a^{\frac{n-1}{2}} \right)^2 = 1$$

e l'equazione $x^2 = 1$ in $\mathbf{Z}_n$ ammette le sole soluzioni $x = 1$ e $x = {}^{-}1$ (questo non è più vero se n è composto; per esempio in $\mathbf{Z}_8$ l'equazione $x^2 = 1$ ammette 4 soluzioni: 1, 3, 5, $7 = {}^{-}1$).

Quindi se n è primo possiamo suddividere i numeri non nulli di $\mathbf{Z}_n$ in due insiemi: quelli che elevati a $(n-1)/2$ danno 1 (si dimostra che sono esattamente la metà di essi), e quelli che elevati a $(n-1)/2$ danno $^{-}1$ (l'altra metà).

Possiamo definire così la **funzione di Legendre**.

Definizione

Sia n un numero primo maggiore di 2 e $a \in \mathbf{Z}_n$. la funzione di Legendre $L(a, n)$ è così definita:

$$L(a, n) := a^{\frac{n-1}{2}} \text{ in } \mathbf{Z}_n.$$

Da quanto abbiamo visto, la funzione di Legendre assume solo tre valori:

$$L(a, n) = \begin{cases} 0 & \text{se } a = 0 \\ 1 & \text{se } a^{\frac{n-1}{2}} = 1 \\ {}^{-}1 & \text{se } a^{\frac{n-1}{2}} = {}^{-}1 \end{cases}$$

Per costruire l'algoritmo di Solovay-Strassen ci serve estendere la funzione di Legendre (che si applica solo a numeri primi) a qualsiasi numero dispari (esattamente come abbiamo fatto per estendere il teorema di Fermat con il teorema di Eulero). Si definisce così la **funzione di Jacobi**.

Definizione

Sia n un numero dispari maggiore di 2, la cui scomposizione in fattori primi è la seguente

$$n = p_1^{k_1} p_2^{k_2} \dots p_t^{k_t},$$

e sia $a \in \mathbf{Z}_n$. La funzione di Jacobi è così definita:

$$J(a,n) := L(a,p_1)^{k_1} L(a,p_2)^{k_2} \dots L(a,p_t)^{k_t}.$$

Per esempio, in $\mathbf{Z}_{21}$ risulta

$$J(2,21) = L(2,3)L(2,7) = {}^{-}1 \cdot 1 = {}^{-}1$$
$$J(3,21) = L(3,3)L(3,7) = 0 \cdot 1 = 0$$
$$J(4,21) = L(4,3)L(4,7) = 1 \cdot 1 = 1$$

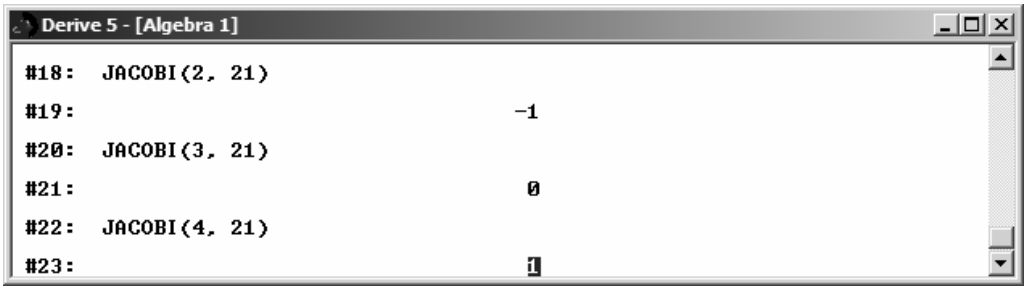
È ovvio che se n è primo allora la funzione di Legendre coincide con la funzione di Jacobi.

Anche la funzione di Jacobi, come la funzione di Legendre, non può che assumere uno dei tre valori 0, $^{-}1$, 1: in particolare risulta

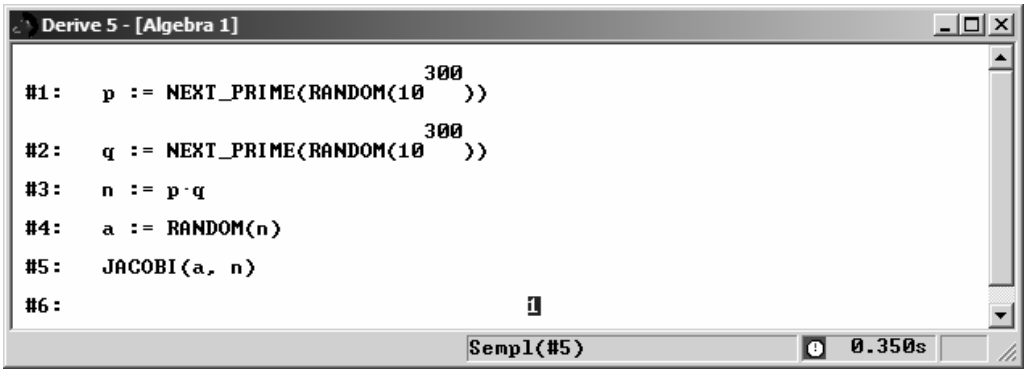
$$J(a,n) = 0$$

se e solo se MCD$(n,a) \neq 1$, cioè se a non è primo con n.

In Derive il comando `JACOBI(a,n)` calcola la funzione di Jacobi.

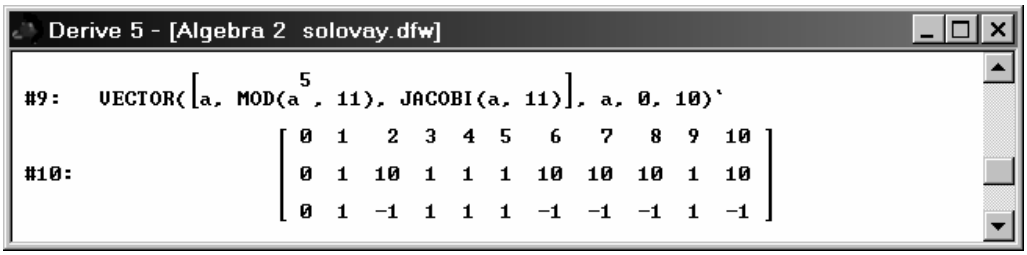


Il fatto interessante è che per calcolare la funzione di Jacobi non occorre conoscere la fattorizzazione di n e per esso esistono algoritmi di calcolo efficienti, sui quali siamo costretti a sorvolare. Con Derive per un numero n di 600 cifre occorrono pochi decimi di secondo.

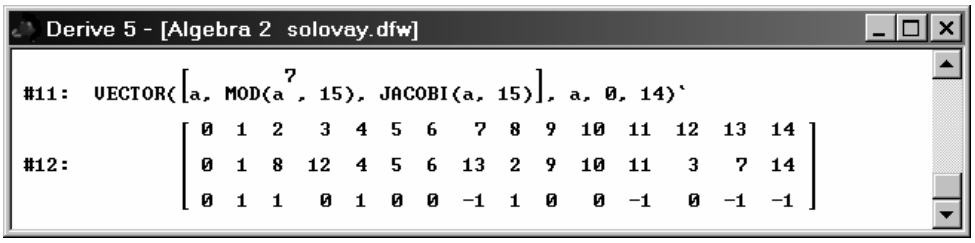


Quindi per un n dispari, ci sono $\varphi(n)$ numeri $a \in \mathbf{Z}_n$ tali che $J(a,n) = {}^{-}1$ oppure $J(a,n) = 1$.

Sappiamo che se n è primo, allora *tutti* gli $a \in \mathbf{Z}_n$ soddisfano l'uguaglianza $a^{\frac{n-1}{2}} = J(a,n)$; la figura seguente esemplifica questa proprietà dei numeri primi con $n = 11$ e mette confronto, per tutti gli $a \in \mathbf{Z}_{11}$ (elencati sulla prima riga), il valore della potenza $a^{\frac{n-1}{2}}$ (seconda riga) con il valore di $J(a,n)$ (terza riga). Ricordiamo che in $\mathbf{Z}_{11}$ risulta $10 = {}^{-}1$.



E se n non è primo? Costruiamo la stessa tabella per un numero dispari composto, per esempio $n = 15$.



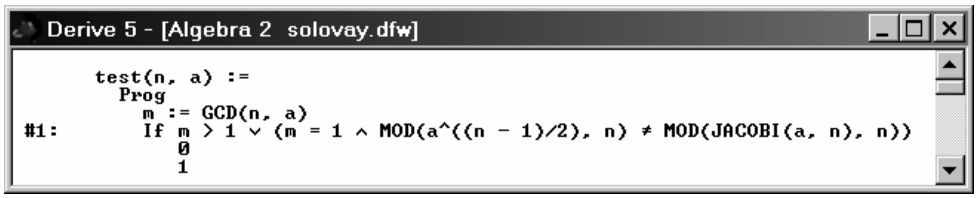
Ignoriamo i numeri non primi con 15 (sono 0, 3, 5, 6, 9, 10, 12, per i quali $J(a, n) = 0$); per gli altri $8 = \varphi(15)$ può risultare:

- $a^{\frac{n-1}{2}} = J(a, n)$; ce ne sono solo 2: 1 e 14; oppure
- $a^{\frac{n-1}{2}} \neq J(a, n)$; ce ne sono 6: 2, 4, 7, 8, 11, 13.

Se $a^{\frac{n-1}{2}} = J(a, n)$ allora a è un **testimone a favore** della primalità di n (si comporta **come se** n fosse primo). Quindi per $n = 15$ ci sono 2 testimoni a favore, e 6 contro; invece per $n = 11$ tutti sono testimoni a favore.

In generale, consideriamo un numero dispari *composto* n e sia $\varphi(n)$ il numero di numeri primi con esso. Di questi, siano $T(n)$ i testimoni a favore, cioè quelli che soddisfano l'uguaglianza $a^{\frac{n-1}{2}} = J(a, n)$, e quindi $\varphi(n) - T(n)$ i testimoni contro, cioè quelli che non la soddisfano.

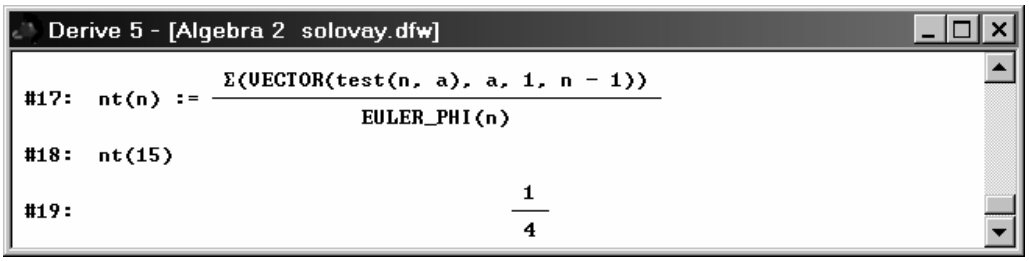
Se si svolge qualche esperimento con diversi numeri composti dispari, si scopre che $T(n)$ è solitamente piccolo; cioè: se n è composto, sono in pochi a testimoniare per la sua primalità. La figura seguente mostra la funzione `test(n,a)`, costruita con Derive, che restituisce 1 se a è un testimone a favore, 0 altrimenti.



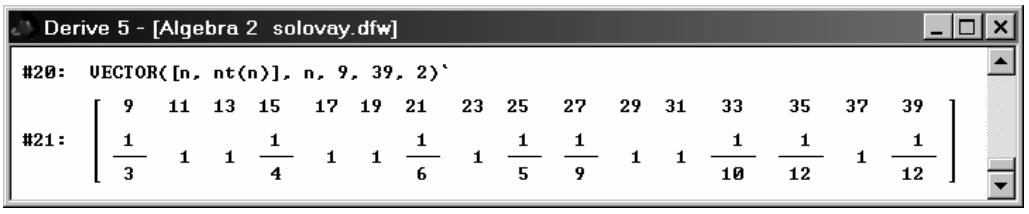
Possiamo così facilmente svolgere qualche esperimento per contare quanti sono, per i numeri composti dispari, i testimoni a favore. La funzione

```
nt(n):=sum(vector(test(n,a),a,1,n-1))/euler_phi(n)
```

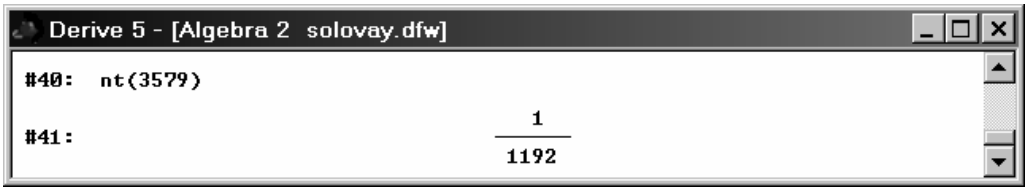
fornisce il rapporto tra $T(n)$ e $\varphi(n)$.



Sfruttiamo la funzione `nt(n)` per valutare quanti sono i testimoni a favore per i numeri dispari da 9 a 39.



Come si vede, il rapporto $T(n)/\varphi(n)$ vale 1 se n è primo, vale in generale molto meno di 1 se n è composto. Per esempio tale rapporto vale solo 1/1192 per $n = 3579$.



In altri termini: se n è primo tutti sono testimoni a favore della primalità di n; se n è composto, i testimoni a favore della primalità di n sono "pochi" (in un senso che tra poco preciseremo).

Gli n che se comportano peggio, da questo punto di vista, sono ancora una volta i numeri di Carmichael: per esempio, per $n = 1729$ risulta $T(n)/\varphi(n) = 1/2$, cioè ben la metà dei numeri primi con n sono pronti a testimoniare a favore della primalità di n. Bene, questo è il caso peggiore che possa capitare.

Poniamoci la stessa domanda che ci siamo posti per il teorema di Fermat: è possibile, pur di aumentare il numero di testimoni a favore, che si possa dichiarare n primo, almeno con una certa **probabilità**? Questa volta la risposta è SI.

Il teorema seguente costituisce la chiave dell'algoritmo di Solovay-Strassen.

Teorema

Sia n un numero *composto* dispari n, $\varphi(n)$ il numero di numeri a primi con n, $T(n)$ il numero di testimoni a per i quali

$$a^{\frac{n-1}{2}} = J(a, n).$$

Allora qualunque sia n risulta

$$\frac{T(n)}{\varphi(n)} \leq \frac{1}{2}.$$

Questo significa che se prendiamo a caso un a primo con n (n composto dispari), la probabilità che a sia un testimone a favore della primalità di n è minore del 50%. Oppure, in altri termini: se per un certo n dispari troviamo un testimone a favore a, scegliendolo a caso tra i $\varphi(n)$ primi con n, allora la probabilità che n sia composto è minore di 1/2. Se di testimoni ne troviamo k, allora la probabilità che n sia composto è minore di

$$\left(\frac{1}{2}\right)^k,$$

il che significa che la probabilità che n sia primo è maggiore di

$$1-\frac{1}{2^k}.$$

Se riusciamo a procurarci 10 testimoni per n, la probabilità che n sia primo è

$$1-\frac{1}{2^{10}} \approx 0.999.$$

Con 20 testimoni la probabilità sale a circa 0.999999. La probabilità che n abbia 50 testimoni (solitamente questo è lo standard di sicurezza) e sia composto è circa di 10^{-15}.

Illustriamo finalmente l'algoritmo di Solovay-Strassen.

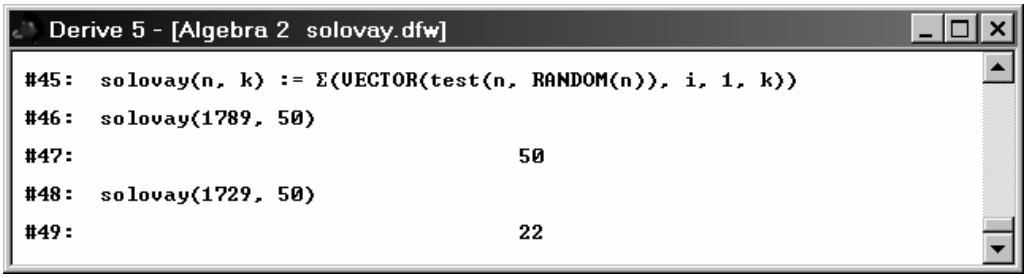
Sia dato un numero dispari n di cui vogliamo stabilire la primalità; si scelga a caso un numero a compreso tra 1 e $n-1$. Allora:

- se MCD$(n, a) > 1$, allora n è certamente composto;
- altrimenti, se $a^{\frac{n-1}{2}} \neq J(a, n)$ allora n è certamente composto;
- altrimenti, se $a^{\frac{n-1}{2}} = J(a, n)$ allora n è **probabilmente primo**, con probabilità maggiore di 1/2.
- si itera il procedimento fino a che: o si conclude che n è composto, oppure si raggiungono 50 testimoni a favore, nel qual caso il numero viene dichiarato primo, con probabilità maggiore di $1-10^{-15}$.

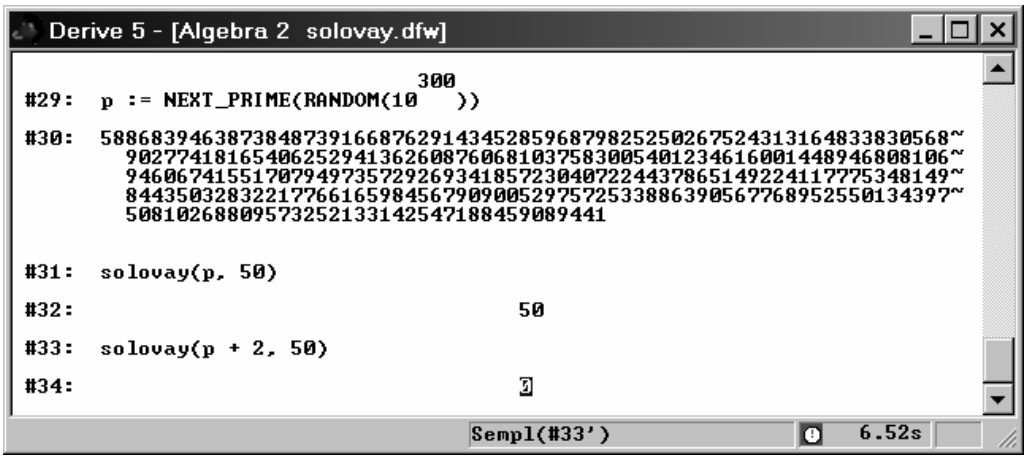
Costruiamo allora la funzione `solovay(n,k)` che fornisce, su k scelte casuali di a compreso tra 1 e $n-1$, il numero di testimoni a favore registrati. Se tale numero è minore di k allora n è certamente composto; se è uguale a k, allora n è primo con probabilità

$$1-\frac{1}{2^k}.$$

La figura seguente mostra il comportamento della funzione `solovay` su 1789 (che è primo) e su 1729 (che è composto), con $k = 50$.



Eseguiamo infine una prova su numeri grandi, di 300 cifre. Derive impiega circa 6 secondi a riconoscere p primo e $p+2$ composto.



Si osservi che per il numero composto, su 50 scelte casuali, non è stato trovato neppure un testimone a favore.

Lo scopo di questa appendice è stato, in definitiva, quello di mostrare come sia possibile *fabbricarsi in casa*, anche con un software non professionale come Derive, numeri primi grandi, di dimensioni utili per definire le chiavi dell'algoritmo RSA.

Bibliografia

Leonesi S., Toffalori C., 2006, "*Numeri e crittografia*", Springer, Italia.

Alberti G., "*Aritmetica finita e crittografia a chiave pubblica*", www.dm.unipi.it/~alberti/files/didattica/divulgazione/crittografia1.pdf

Michele Impedovo