Apparso in "Ricordando Franco Conti",
Scuola Normale Superiore, Pisa 2004.

[versione: 15 dicembre 2004]

Aritmetica finita e crittografia a chiave pubblica
Un percorso didattico per studenti delle Scuole Medie Superiori

Giovanni Alberti

Introduzione

La presente è una versione opportunamente editata degli appunti di una serie di lezioni tenute nel 2001 e ripetute nel 2004 nell'ambito di un ciclo di seminari divulgativi organizzati con cadenza annuale dal Dipartimento di Matematica dell'Università di Pisa, e rivolto agli studenti degli ultimi due anni delle scuole medie superiori della zona.

L'obiettivo di queste lezioni era spiegare agli studenti alcuni risultati elementari di aritmetica modulare, e mostrarne l'utilizzazione nel campo della crittografia. Dopo una prima lezione introduttiva dedicata a crittografia e codici a chiave pubblica, gli argomenti delle successive sono stati: nozioni di base di aritmetica modulare, definizione e stima del costo computazionale di un algoritmo (in una versione opportunamente semplificata), algoritmo di Euclide per la ricerca del massimo comun divisore di due numeri interi, piccolo teorema di Fermat; ho concluso la quarta lezione illustrando il codice a chiave pubblica RSA.

Ho scelto questo argomento in primo luogo perché costituisce un'interessante combinazione di matematica a prima vista molto astratta e di applicazioni assai concrete; quindi anche un'occasione per sottolineare che la matematica non si divide in astratta da una parte ed applicata dall'altra.

Dal punto di vista didattico questa scelta presenta diversi vantaggi: la matematica necessaria per capire il funzionamento del codice RSA a livello non superficiale può essere spiegata ad uno studente delle scuole medie ben intenzionato in poche ore; inoltre, finalizzando l'esposizione alla risoluzione di un problema, cioè quello di trovare un codice a chiave pubblica, è possibile introdurre in modo naturale un'ampia varietà di risultati: dal piccolo teorema di Fermat alle stime del costo computazionale dell'algoritmo di Euclide.

Nella preparazione delle lezioni ho presupposto un uditorio a suo agio solo con pochi concetti di aritmetica pre-universitaria (numeri primi, fattorizzazione dei numeri interi, massimo comun divisore), con un minimo di terminologia insiemistica (funzioni e insiemi) e che si ricordasse gli algoritmi per fare le operazioni aritmetiche elementari a mano. Queste note si discostano dal percorso svolto nelle lezioni solo occasionalmente: alcune dimostrazioni sono presentate

in maggior dettaglio, e alla fine di ogni capitolo ho aggiunto una lista di esercizi, alcuni dei quali sviluppano punti appena accennati nel testo (e possono essere svolti solo con l'aiuto di un insegnante). Insisto su dimostrazioni ed esercizi nella convinzione che siano ingredienti essenziali per una comprensione non superficiale di qualunque argomento matematico.

Bilbiografia: per la parte matematica di queste lezioni, mi sono avvalso del classico testo di Neal Koblitz [1]; pur essendo inteso per corsi universitari avanzati, la parte iniziale è di fatto accessibile ad uno studente con buone basi di matematica pre-universitaria. La quasi totalità delle informazioni storiche ed extra-matematiche provengono dal libro di Simon Singh [2]: si tratta di un testo divulgativo sulla storia della crittografia, scritto in modo avvincente ed accessibile a chiunque; ad esso devo l'interesse che ho per l'argomento.

Infine vorrei sottolineare che né la crittografia né l'aritmetica sono in alcun modo vicine ai miei interessi di ricerca; mi scuso quindi per le imprecisioni ed ingenuità che avrò inevitabilmente disseminato in queste note.

1. Crittografia e crittografia a chiave pubblica

La situazione standard della crittografia è la seguente: una persona A deve far arrivare un messaggio dal contenuto riservato ad una persona B, e teme che una terza persona C lo possa intercettare: ad esempio, il quartier generale di un esercito (A) deve comunicare le istruzioni ad una determinata unità (B) sapendo che il nemico (C) sta sicuramente ascoltando le trasmissioni radio. Una possibile soluzione è che A manipoli il messaggio prima di trasmetterlo, seguendo una certa procedura nota solo a lui e a B, cosicché, quand'anche C venisse in possesso del messaggio manipolato, non sia in grado di ricostruirne il significato.

La manipolazione fatta da A per rendere il messaggio inintelligibile ad occhi estranei si chiama *crittazione*, mentre l'operazione inversa, quella fatta da B per ricostruire il messaggio originale, si chiama *decrittazione*. La *crittografia* è lo studio dei *codici*, ovvero delle procedure di crittazione e decrittazione utilizzate da A e B, come pure delle tecniche adoperate da C per arrivare a scoprire il codice usato da A e B.

1.1. Schema generale

Molte procedure di crittazione possono essere schematizzate come segue:

(i) A scrive il testo da spedire come una sequenza continua di caratteri[1] che poi divide in blocchi di m caratteri consecutivi, con m concordato con B;

[1] Per caratteri si intendono le lettere dell'alfabeto ed eventualmente le cifre da `0` a `9` e lo spazio vuoto che separa la parole; si può decidere di scrivere tutto il testo in minuscole, oppure di distinguere minuscole e maiuscole, nel qual caso `A` e `a` sono caratteri distinti; si possono inserire o meno i caratteri di interpunzione (punti, virgole, ecc.) o altri simboli (`$`, §, `&`, ecc.). La lunghezza del messaggio è il numero di caratteri utilizzati per scriverlo.

(ii) A ottiene il testo crittato sostituendo ad ogni blocco del testo originale un altro; questa "traduzione" viene fatta utilizzando una specie di dizionario, in possesso sia di A che di B, che ad ogni possibile blocco di m caratteri associa univocamente un nuovo blocco di m caratteri;

(iii) A spedisce a B solo il testo crittato;

(iv) B scompone il messaggio crittato in blocchi di m caratteri, ed usa il dizionario all'inverso per trovare il significato originale di ciascun blocco.

Per esempio, A divide il testo da trasmettere `baci␣e␣abbracci` in blocchi di 4 caratteri, [`baci`], [`␣e␣a`], [`bbra`],[`cci␣`], e consultando la sezione *Noncrittato*→*Crittato* del "dizionario", trova che [`baci`] va sostituito con [`sjwr`], [`␣e␣a`] con [`rpp3`], [`bbra`] con [`3␣ew`], e [`cci␣`] con [`zzzz`]. Il messaggio crittato è quindi `sjwrrpp33␣ewzzzz`, e per decrittarlo, B consulterà la sezione *Crittato*→*Noncrittato* del "dizionario" alle voci [`sjwr`], [`rpp3`], [`3␣ew`] e [`zzzz`].

Questo schema è però un'astrazione, ed un codice costruito in questo modo non sarebbe praticamente utilizzabile: se ad esempio si decide di scomporre i messaggi in blocchi di 4 caratteri, limitati per semplicità alle 21 lettere dell'alfabeto italiano allo spazio vuoto, per un totale di 22 diversi caratteri, i possibili blocchi sono $22^4 = 234256$; il dizionario dovrebbe quindi contenere oltre duecentomila voci in entrambe le sezioni, e finirebbe per contare diverse centinaia di pagine. Non solo le procedure di crittazione e decrittazione sarebbero farraginose, ma cambiare il dizionario con frequenza periodica—precauzione necessaria nel caso che venga usato da una vasta rete di comunicazioni come ad esempio quella di un esercito—risulterebbe impraticabile.

1.2. Un esempio realistico: il codice di Vigenère

Storicamente si sono invece affermati codici che per tradurre i blocchi usano formule relativamente semplici. Un classico esempio è il codice di Vigenère: A e B stabiliscono di dividere i messaggi in blocchi di 4 caratteri, e poi concordano 4 numeri interi, ed esempio $5, 1, 7, 3$; quando A deve crittare un blocco di 4 lettere, sostituisce la prima con quella che si trova 5 posti *a destra* nell'ordine alfabetico, la seconda con quella 1 posto a destra, la terza con quella 7 posti a destra, ed infine la quarta con quella 3 posti a destra;[2] così facendo il messaggio `baci␣e␣abbracci␣` diventa `gblnefgdgcadhdrc`. Per decrittare il messaggio, B ripete la stessa operazione, sostituendo alla prima lettera di ciascun blocco quella 5 posti *a sinistra* nell'ordine alfabetico, e così via. Più in generale, A e B concordano la lunghezza m dei blocchi ed una sequenza di numeri interi $n_1, n_2, \ldots, n_m$. L'unica cosa che A e B devono ricordare e tenere accuratamente segreta è la sequenza $n_1, n_2, \ldots, n_m$, chiamata appunto *chiave di crittazione*; per incrementare la sicurezza delle comunicazioni è sufficiente cambiare la chiave con frequenza periodica.

[2] Nel fare questo si conviene che lo spazio vuoto ␣ costituisce la prima lettera dell'alfabeto, e che una volta arrivati alla `z` si ricomincia dall'inizio dell'alfabeto, per cui, ad esempio, la quarta lettera a destra della `v` è la `b`. Per inserire altri simboli tra i caratteri utilizzabili, bisogna assegnargli un posto nell'ordine alfabetico.

Perfezionato da Blaise de Vigenère verso la fine del XVI secolo, questo codice è stato considerato a lungo indecifrabile; solo nel 1863 Friedrich Wilhelm Kasiski pubblicò un metodo per ricavare la chiave di crittazione a partire da un messaggio crittato di sufficiente lunghezza.

Si noti che, prima di Kasiski, la sicurezza di tale codice non dipendeva dalla segretezza principio di funzionamento—che era anzi noto a tutti gli esperti di crittografia—bensì dalla segretezza della chiave. Questo è un punto importante: un buon codice è tale se la sua sicurezza non dipende dalla segretezza del principio di funzionamento quanto da quella di un certo insieme di parametri a scelta di A e B, detti appunto chiavi.

1.3. Un po' di matematica

La possibilità di usare calcolatrici meccaniche e poi computer ha portato rapidamente all'introduzione di codici basati su formule matematiche. Per illustrarne lo schema di funzionamento generale abbiamo bisogno di alcune definizioni: fissato un numero intero n maggiore di 1, indichiamo con $\mathbb{Z}_n$ (leggi: "zeta enne" oppure "zeta modulo enne") l'insieme dei numeri interi compresi tra 0 ed $n-1$, vale a dire

$$\mathbb{Z}_n := \{0, 1, 2, \dots, n-1\} \ .$$

Ricordo che una *funzione* f da $\mathbb{Z}_n$ in $\mathbb{Z}_n$ è una qualunque procedura che assegni ad ogni numero x appartenente a $\mathbb{Z}_n$ un nuovo numero, indicato appunto con $f(x)$, appartente sempre a $\mathbb{Z}_n$. La *funzione inversa* di f, indicata con f^{-1}, è quella che disfa quanto f ha fatto, ovvero che associa a ciascun $y \in \mathbb{Z}_n$ esattamente in quell'x tale che $f(x) = y$. In altre parole, f ed f^{-1} sono collegate dalla relazione

$$f^{-1}(f(x)) = x \quad \text{per ogni } x \in \mathbb{Z}_n.$$

Per esempio, dare una funzione f da $\mathbb{Z}_7$ in $\mathbb{Z}_7$ significa assegnare ad ogni numero intero compreso tra 0 e 6 un valore $f(x)$ compreso anch'esso tra 0 e 6, come illustrato nella tabella seguente:

x:	0	1	2	3	4	5	6
$f(x)$:	5	1	4	0	3	6	2

(1.1)

Il valore di $f^{-1}(6)$ è per definizione quel numero x tale che $f(x) = 6$; cercando il 6 nella seconda riga della tabella (1.1) otteniamo che questo corrisponde ad $x = 5$, e quindi $f^{-1}(6) = 5$. Procedendo a questo modo possiamo con un po' di pazienza compilare la tabella della funzione inversa f^{-1}:

y:	0	1	2	3	4	5	6
$f^{-1}(y)$:	3	1	6	4	2	0	5

(1.2)

Ovviamente è possibile compilare questa tabella perché ad ogni valore di y corrisponde un solo valore di x tale che $f(x) = y$. Quest'ultima osservazione corrisponde al fatto ben noto che, dato un insieme X con un numero finito di elementi, una funzione f da X in sé è invertibile (cioè esiste la funzione inversa f^{-1}) se e solo se f è *iniettiva*, vale a dire che assegna valori diversi a numeri diversi (cfr. esercizio 1.20).

1.4. Codici a base numerica

Il principio di funzionamento generale dei codici che ci interessa esaminare può essere schematizzato come segue:

(i) A usa un qualche procedimento standard (non segreto) per tradurre il messaggio in una sequenza di numeri $x_1, x_2, x_3, \ldots$ appartenenti a $\mathbb{Z}_n$, dove n è un numero intero concordato con B;

(ii) A associa a ciascun numero x_i un nuovo numero y_i utilizzando una funzione $f : \mathbb{Z}_n \to \mathbb{Z}_n$ concordata con B, vale a dire calcola $y_i := f(x_i)$ per ogni i;

(iii) A trasmette a B la sequenza crittata $y_1, y_2, y_3, \ldots$;

(iv) B determina i numeri della sequenza originale a partire da quelli della sequenza crittata utilizzando la funzione inversa f^{-1}, ovvero calcola $x_i = f^{-1}(y_i)$ per ogni i;

(v) B ritraduce la sequenza $x_1, x_2, x_3, \ldots$ in lettere e può finalmente leggere il messaggio.

Osservazioni – 1. Il fatto che la funzione f debba essere iniettiva è ancora più chiaro nel contesto dei codici: se ci fossero due numeri distinti x ed x' in $\mathbb{Z}_n$ a cui f assegna lo stesso valore y, qualora B incontrasse il numero y nella sequenza crittata speditagli da A non avrebbe alcun modo per decidere se la sequenza originale contiene x oppure x', e quindi non sarebbe in grado di ricostruirla.

2. La conversione del messaggio da sequenza di caratteri a sequenza di numeri in $\mathbb{Z}_n$ nel passo (i) e la sua riconversione a sequenza di caratteri nel passo (v) sono fatte seguendo procedimenti standard tutt'altro che segreti, e non hanno alcuna rilevanza ai fini dell'efficacia del codice; pertanto non ci dilungheremo su queste operazioni. La crittazione e decrittazione vere e proprie hanno luogo nei passi (ii) ed (iv); quelle che vanno tenute segrete sono le funzioni f ed f^{-1}, che chiameremo rispettivamente *chiave di crittazione* e *chiave di decrittazione*.

1.5. Perché usare sequenze di numeri?

Riflettendo un attimo ci si accorge che lo schema descritto in Sezione 1.4 non è concettualmente diverso da quello descritto in Sezione 1.1; la tabella corrispondente alla funzione f fa le veci della sezione *Noncrittato*→*Crittato* del dizionario, mentre la tabella corrispondente alla funzione inversa f^{-1} fa le veci della sezione *Crittato*→*Noncrittato*.

Se ci limitassimo a descrivere le funzioni tramite tabelle come la (1.1) non ci sarebbe alcuna differenza significativa tra le due impostazioni. Tuttavia, un modo alternativo per costruire funzioni, anzi il modo che tutti noi consideriamo più naturale, è usare formule matematiche. Come vedremo, è proprio questa possibilità che rende vantaggioso lo schema di codice descritto in Sezione 1.4.

C'è però un problema: per costruire formule si usano solitamente operazioni algebriche elementari come somma e prodotto; ovviamente noi possiamo sommare e moltiplicare i numeri in $\mathbb{Z}_n$, ma il risultato di queste operazioni potrebbe non essere più un numero in $\mathbb{Z}_n$. Ad esempio per $n = 6$ la formula $f(x) := 2x + 1$ assegna ai numeri $0, 1, 2$ valori compresi tra 0 e 5, ma assegna a $3, 4, 5$ valori maggiori di 5, e quindi non definisce una funzione da $\mathbb{Z}_6$ in $\mathbb{Z}_6$. Per ovviare a questa difficoltà bisogna ridefinire il significato di somma e prodotto di numeri in $\mathbb{Z}_n$ in modo tale che il risultato di queste operazioni sia sempre un numero in $\mathbb{Z}_n$. Nel prossimo capitolo spiegheremo come.

1.6. Crittografia a chiave pubblica

Se acquistiamo un qualche prodotto via internet, solitamente lo paghiamo usando la nostra carta di credito, ovvero digitando al momento giusto il numero della carta stessa. Siccome il collegamento tra il nostro computer (A) ed il server della compagnia da cui stiamo acquistando (B) è tutt'altro che diretto, ma passa anzi attraverso molti intermediari a partire dal provider a cui siamo collegati, ogni messaggio inviato da A a B è facilmente intercettabile da terzi, e bisogna quindi crittare i messaggi da A a B in modo tale che nessun malintenzionato (C) possa leggere il numero di carta di credito. Il problema è che A e B non si sono preventivamente messi d'accordo su un codice da usare a questo scopo.

La procedura utilizzata in queste situazioni segue questo schema astratto: B trasmette ad A la sezione *Noncrittato*→*Crittato* del dizionario da utilizzare, A critta il messaggio seguendo i passi descritti in Sezione 1.1, e lo spedisce a B che lo decritta usando la sezione *Crittato*→*Noncrittato* del dizionario.

A prima vista questo sistema non sembra poter funzionare: se C intercetta il messaggio iniziale da B ad A contenente il dizionario—messaggio che ovviamente non può essere crittato—allora dovrebbe essere in grado di decifrare ogni messaggio spedito da A a B.

Il punto è che B spedisce ad A solo la sezione *Noncrittato*→*Crittato* del dizionario, cioè quella che si usa nella fase di crittazione, ma non la sezione *Crittato*→*Noncrittato*, che sarebbe quella di cui C ha bisogno per leggere i messaggi di A. Chiaramente C può ricostruire la sezione *Crittato*→*Noncrittato* del dizionario a partire dalla sezione *Noncrittato*→*Crittato*, ma questo richiede tempo.[3] Questo è il punto: se l'operazione di ricostruzione della parte mancante

[3] C si trova nella situazione dello studente che deve tradurre un testo tedesco in italiano avendo a disposizione solo la sezione *Italiano*→*Tedesco* del dizionario: se incontra una parola tedesca di cui non riesce ad indovinare il corrispondente italiano, l'unica cosa che può fare è scorrere la metà

del dizionario richiede un tempo enorme, C non può decrittare i messaggi di A, e quindi il codice è sicuro.

Traduciamo tutto ciò nello schema descritto in Sezione 1.4: B spedisce ad A la chiave di crittazione f senza fare alcuno sforzo per tenerla nascosta ad occhi indiscreti; A critta il messaggio e lo spedisce a B; B decritta il messaggio usando la chiave di decrittazione f^{-1}, che si è ben guardato dal divulgare. La funzione f viene chiamata *chiave pubblica* proprio perché potenzialmente accessibile a tutti, mentre f^{-1} viene chiamata *chiave privata*. In teoria è sempre possibile ricavare la chiave privata f^{-1} a partire da quella pubblica f,[4] ma questa operazione può essere impossibile in pratica perché richiede troppo tempo.

Un codice basato su questo principio è detto *codice a chiave pubblica*.

Bisogna dire a questo punto che non è affatto evidente che un simile codice sia realizzabile, visto tra l'altro che la ricostruzione della chiave privata a partire da quella pubblica non verrebbe certo fatta a mano ma tramite l'ausilio di un computer. Ed infatti, quando nel 1975 W. Diffie, M. Hellman e R. Merkle formularono per la prima volta il concetto di codice a chiave pubblica, non riuscirono a dare esempi concreti; solo nel 1977, R. Rivest, A. Shamir e L. Adleman proposero un codice con la caratteristiche richieste; questo codice, noto come RSA dalle iniziali dei loro cognomi, è tutt'oggi uno dei più utilizzati.

1.7. Lo scambio delle chiavi

L'ambito di applicazione dei codici a chiave pubblica non riguarda solamente la sicurezza delle transazioni economiche via internet, ma quella di tutte le comunicazioni informatiche (e non solo). In effetti esistono da tempo codici non a chiave pubblica considerati relativamente sicuri—uno dei più comunemente usati è il *Data Encryption Standard*, o DES. Tuttavia, se A e B intendono usare uno di questi codici per rendere sicure le loro comunicazioni, devono prima trovare un modo sicuro di comunicarsi le chiavi. Ma trovarne uno che non implichi incontrarsi di persona o spedire lettere sigillate è un problema più complicato di quanto non possa sembrare a prima vista. La soluzione oggi più usata è quella proposta sempre da Diffie, Hellman e Merkle nel 1975, ed utilizza, tra gli altri ingredienti, un codice a chiave pubblica.

Complementi ed esercizi

1.8. Rappresentazione di un numero in base n – La rappresentazione di un numero intero positivo x in base n è una sequenza di numeri interi

---

del dizionario a sua disposizione finché non la incontra. Pertanto C procede molto più lentamente di A, che deve invece tradurre dall'italiano al tedesco avendo a disposizione la metà giusta del dizionario.

[4] In mancanza di soluzioni migliori, C può comunque compilare la tabella dei valori di f e procedere come descritto in Sezione 1.3 per ricavare la tabella di f^{-1}.

$[x_k, x_{k-1}, \dots x_1, x_0]$ compresi tra 0 ed $n-1$, detti appunto le cifre di x in base n,[5] e caratterizzati dalla seguente formula:

$$x = x_k \cdot n^k + x_{k-1} \cdot n^{k-1} + \cdots + x_2 \cdot n^2 + x_1 \cdot n + x_0 \ . \tag{1.3}$$

Il significato di questa formula è evidente per $n = 10$: se la rappresentazione decimale di x è 736 allora

$$x = 7 \cdot 10^2 + 3 \cdot 10^1 + 6 \cdot 10^0 \ .$$

1.9. – Sia x il numero rappresentato in base 22 dalla sequenza di cifre $[1, 21, 3, 17]$. Usare la formula (1.3) per determinarne la rappresentazione in base decimale.

1.10. – Dato un numero reale a, indichiamo con $\lceil a \rceil$ il più piccolo numero intero strettamente maggiore di a. Utilizzando la formula (1.3), dimostrare che il numero di cifre della reppresentazione di x in base n è

$$k = \left\lceil \frac{\log x}{\log n} \right\rceil$$

dove log sta per il logaritmo in base 10 (si ricordi che $\log x / \log n$ è uguale al logaritmo di x in base n).

1.11. Conversione dalla base 10 alla base n – Dato un numero intero positivo x, come si trovano le cifre $[x_k, x_{k-1}, \dots x_1, x_0]$ della sua rappresentazione in base n? Come si vede dalla formula (1.3), dividendo x per n otteniamo come resto x_0 e come quoziente

$$d_0 := x_k \cdot n^{k-1} + x_{k-1} \cdot n^{k-2} + \cdots + x_2 \cdot n + x_1 \ .$$

Dividendo il quoziente d_0 per n otteniamo come resto x_1. Dividendo il quoziente della precedente divisione per n otteniamo come resto x_2, ecc. Procediamo a questo modo finché non otteniamo come quoziente 0.

1.12. – Usare l'algoritmo descritto in Sezione 1.11 per trovare la rappresentazione in base 22 di $x = 536$ (rappresentazione decimale).

1.13. – L'algoritmo di conversione descritto in Sezione 1.11 richiede di fare divisioni con resto di numeri interi. Trovare il modo di farle usando una comune calcolatrice tascabile.

1.14. Conversione di un messaggio in sequenza di numeri – Tradurre un messaggio formato da una sequenza di caratteri in una sequenza di

[5] Si noti però che per la rappresentazione in base 16 si usano comunemente i numeri da 0 a 9 seguiti dalle lettere da A ad F, e non i numeri da 0 a 15.

numeri in $\mathbb{Z}_n$ può essere visto come una conversione a base n. Se ad esempio l'insieme dei caratteri ammissibili sono le 21 lettere dell'alfabeto più lo spazio vuoto, possiamo identificarli con i numeri da 0 a 21 (lo 0 corrisponde allo spazio vuoto $\sqcup$, l'1 alla lettera `a`, il 2 alla `b` e così via) ed il messaggio può quindi essere visto com un numero intero scritto in base 22. A questo punto è possibile riscriverlo in base decimale usando la formula (1.3) e poi convertirlo in base n, ovvero scriverlo come sequenza di numeri in $\mathbb{Z}_n$, usando il procedimento descritto in Sezione 1.11.

1.15. – Verificare che per $n = 22$ la procedura descritta in Sezione 1.14 per convertire un messaggio in una sequenza di numeri in $\mathbb{Z}_{22}$ equivale alla seguente: ad ogni lettera si sostituisce il numero corrispondente alla posizione nell'ordine alfabetico, e allo spazio vuoto $\sqcup$ il numero 0.

1.16. – Verificare che per n uguale a 22^m, la procedura descritta in Sezione 1.14 equivale alla seguente: si divide il messaggio in blocchi di m caratteri, e ad ogni blocco si sostituisce il numero

$$x := x_1 \cdot 22^{m-1} + x_2 \cdot 22^{m-2} + \ldots + x_{m-1} \cdot 22 + x_m \ ,$$

dove x_1 è il numero corrispondente alla posizione della prima lettera del blocco, x_2 quello della seconda, ecc.

1.17. – Si consideri un messaggio di lunghezza h scritto nel solito alfabeto di 22 caratteri. Dimostrare che la conversione descritta in Sezione 1.14 produce una sequenza di numeri in $\mathbb{Z}_n$ di lunghezza k pari a

$$k = \left\lceil \frac{h \log 22}{\log n} \right\rceil$$

(cfr. esercizio 1.10). Come va modificata questa formula se il messaggio è originariamente scritto in un alfabeto di 256 caratteri?

1.18. – Usando la procedura descritta in Sezione 1.14, scrivere il messaggio `baci␣e␣abbracci` come numero in base 10 e poi in base 7.

1.19. – Eseguire tutti i passaggi dello schema di crittazione e decrittazione descritto in Sezione 1.4 per il messaggio `baci␣e␣abbracci`, prendendo $n = 7$ ed f la funzione nella tabella (1.1).

1.20. – Siamo X ed Y due insiemi finiti con lo stesso numero di elementi, ed f una funzione che associa ad ogni elemento di X un elemento di Y, cioè $f : X \to Y$. Verificare che le seguenti proprietà sono equivalenti:

(a) f è iniettiva, cioè $x \neq x' \Rightarrow f(x) \neq f(x')$;

(b) f è suriettiva, cioè l'insieme dei valori assunti da f è tutto Y;

(c) f è invertibile.

2. Aritmetica in $\mathbb{Z}_n$: definizioni e proprietà elementari

Dato un numero intero n maggiore di 1, indichiamo come come già detto con $\mathbb{Z}_n$ l'insieme

$$\mathbb{Z}_n := \{0, 1, 2, \dots, n-1\} .$$

2.1. Somma e prodotto di numeri in $\mathbb{Z}_n$

Dobbiamo pensare a $\mathbb{Z}_n$ come un insieme ciclico in cui a 0 segue 1, a 1 segue 2 e così via fino a $n-1$, a cui segue di nuovo 0, per cui la somma di $n-1$ e 1 non è n (che non appartiene a $\mathbb{Z}_n$) bensì 0. Più in generale, la somma di due numeri a e b in $\mathbb{Z}_n$ corrisponde a quella solita se risulta minore di n, e altrimenti gli si sottrae n. Ad esempio, la somma di 5 e 4 in $\mathbb{Z}_7$ non è 9, ma $9 - 7 = 2$.

Questo curioso modo di fare le somme viene utilizzato in almeno una situazione della vita di tutti i giorni: per le ore dell'orologio. Se ad esempio alle dieci dite a qualcuno di richiamarvi dopo quattro ore, vi aspettate che si faccia vivo verso le due. Il conto che avete fatto è la somma 10 più 4 in $\mathbb{Z}_{12}$.

Una via leggermente diversa, e per certi versi più conveniente, di definire la somma di numeri in $\mathbb{Z}_n$, è di introdurre la nozione di congruenza modulo n: due numeri interi (positivi o negativi) a e b si dicono *congruenti modulo* n se differiscono per un multiplo di n, ovvero se n divide la differenza $a - b$, ed in tal caso scriviamo

$$a \equiv b \mod n.$$

Ogni numero intero a è congruente modulo n ad uno ed un solo numero appartenente a $\mathbb{Z}_n$, e cioè il resto della divisione di a per n;[6] chiamiamo *riduzione* l'operazione che associa ad a tale numero. Per sommare due numeri in $\mathbb{Z}_n$, ne calcoliamo la somma nel modo usuale e poi la riduciamo a $\mathbb{Z}_n$, e cioè prendiamo il numero in $\mathbb{Z}_n$ ad essa congruente. Ad esempio

$$\begin{aligned}
&7 + 6 \equiv 13 \equiv 3 && \mod 10,\\
&75 + 96 \equiv 171 \equiv 71 && \mod 100,\\
&5 + 4 \equiv 9 \equiv 3 && \mod 6,\\
&115 + 120 \equiv 235 \equiv 4 && \mod 11.
\end{aligned}$$

La moltiplicazione di due numeri in $\mathbb{Z}_n$ si definisce in modo analogo: si prende il prodotto nel senso usuale e poi lo si riduce a $\mathbb{Z}_n$. Ad esempio

$$\begin{aligned}
&7 \cdot 6 \equiv 42 \equiv 2 && \mod 10,\\
&75 \cdot 96 \equiv 7200 \equiv 0 && \mod 100,\\
&5 \cdot 4 \equiv 20 \equiv 2 && \mod 6,\\
&115 \cdot 120 \equiv 13800 \equiv 6 && \mod 11.
\end{aligned}$$

Siccome la somma ed il prodotto di numeri interi soddisfano la proprietà commutativa, associativa e distributiva, non è difficile verificare che le stesse proprietà valgono anche per somma e prodotto di numeri in $\mathbb{Z}_n$.

[6] Se la divisione di a per n dà come risultato d con resto r, allora $a = dn + r$, e quindi a è congruo ad r modulo n; inoltre r è compreso tra 0 ed $n - 1$.

2.2. Opposto di un numero in $\mathbb{Z}_n$

Se sommiamo a ed $n-a$ in $\mathbb{Z}_n$ otteniamo 0. Pertanto $n-a$ è l'*opposto* di a in $\mathbb{Z}_n$, e viene talvolta indicato con $-a$. La nozione di opposto ci permette di inquadrare la differenza come caso particolare di somma: $b-a$ non è altro che la somma di b e $-a$.

2.3. Inverso di un numero in $\mathbb{Z}_n$

Diciamo che un certo numero è l'*inverso* di a in $\mathbb{Z}_n$ se moltiplicandolo per a si ottiene 1. Indichiamo tale numero, se esiste, con il simbolo a^{-1}. Chiaramente 0 non ha inverso, mentre 1 e -1 (ovvero $n-1$) sono gli inversi di se stessi.

Per gli altri numeri l'esistenza dell'inverso dipende da n: ad esempio, in $\mathbb{Z}_7$, l'inverso di 2 è 4, perché $2 \cdot 4 \equiv 8 \equiv 1 \bmod 7$ mentre in $\mathbb{Z}_6$ il numero 2 non ha inverso; questo lo possiamo verificare facendo tutti i tentativi possibili (non sono molti) oppure osservando che moltiplicando 2 per un qualunque altro numero si ottiene sempre un numero pari, mentre tutti i numeri congruenti ad 1 modulo 6, e cioè $7, 13, 19, \ldots$, sono dispari. Con un po' di pazienza si vede che in $\mathbb{Z}_7$ tutti i numeri tranne lo 0 hanno un inverso, mentre in $\mathbb{Z}_6$ i soli numeri che hanno un inverso sono 1 e 5 (cioè -1).

Teorema 2.4 – *Un numero a in $\mathbb{Z}_n$ ha un inverso se e solo se il massimo comun divisore tra a ed n, che indichiamo con* $\mathrm{MCD}(a;n)$*, è uguale a* 1*, ovvero se a è primo con n.*

Dimostrazione – Supponiamo per cominciare che $\mathrm{MCD}(a;n) = 1$. Consideriamo i multipli di a

$$0,\ a,\ 2a,\ 3a, \ldots,\ (n-1)a,$$

a prendiamone le riduzioni in $\mathbb{Z}_n$. I numeri così ottenuti sono tutti diversi tra loro[7] e dunque tanti quanti gli elementi di $\mathbb{Z}_n$. Pertanto almeno uno di loro deve essere 1, ovvero esiste un multiplo di a congruo a 1 modulo n, e quindi a è invertibile.

Viceversa, supponiamo che a ed n abbiano un fattore comune $p \neq 1$. Allora tutti i multipli di a sono divisibile per p. D'altra parte, i numeri congrui ad 1 modulo n non sono mai divisibili per p,[8] e pertanto a non può avere un inverso. □

Osservazioni – 1. Se p è un numero primo, allora ogni numero $a \in \mathbb{Z}_p$ diverso da 0 è primo con p, e quindi ammette un inverso. Dunque $\mathbb{Z}_p$ è quel

[7] Per la precisione, si dimostra che dati k, h tali che $0 \leq h < k < n$, allora ka ed ha non sono congruenti modulo n. Infatti, se per assurdo lo fossero, allora n dividerebbe il prodotto $(k-h)a$, e siccome per ipotesi n non ha alcun fattore in comune con a, dovrebbe necessariamente dividere $k-h$. Ma questo è impossibile perché $k-h$ è minore di n e non è 0.

[8] Sono infatti numeri della forma 1 più un multiplo di n, che è anche un multiplo di p, e dividendoli per p il resto è sempre 1.

che si dice un *campo*. Se invece n non è primo esistono sempre numeri in $\mathbb{Z}_n$ diversi da 0 che non ammettono inverso (ad esempio i fattori propri di n). In questo caso $\mathbb{Z}_n$ non è un campo ma solamente un *anello*.

2. Il fatto che a abbia o meno un inverso dipende da a ma anche da n: come abbiamo visto, il numero 2 ha un inverso in $\mathbb{Z}_7$ ma non in $\mathbb{Z}_6$. Non solo, ma l'inverso stesso dipende da n: l'inverso di 2 in $\mathbb{Z}_7$ è 4, mentre in $\mathbb{Z}_5$ è 3.

3. La dimostrazione del Teorema 2.4 non fornisce un vero e proprio metodo per trovare l'inverso di a oltre al puro e semplice procedere per tentativi, e cioè calcolare tutti i multipli di a finché non si ottiene 1. Ad esempio, per trovare l'inverso di 19 in $\mathbb{Z}_{25}$, che ci deve essere perché $\text{MCD}(19;25) = 1$, facciamo i seguenti tentativi $19 \cdot 1 \equiv 19$; $19 \cdot 2 \equiv 38 \equiv 13$; $19 \cdot 3 \equiv 57 \equiv 7$; $19 \cdot 4 \equiv 76 \equiv 1$ e dunque $19^{-1} = 4$. Se però cerchiamo l'inverso di 6 in $\mathbb{Z}_{25}$ dobbiamo fare una ventina di moltiplicazioni prima di scoprire che $6 \cdot 21 \equiv 126 \equiv 1$.

2.5. Una classe di funzioni invertibili

Per costruire codici servono funzioni invertibili da $\mathbb{Z}_n$ in sé (cfr. Sezione 1.4). Il Teorema 2.4 ci permette costruirne molte: presi infatti $a, b \in \mathbb{Z}_n$ con $\text{MCD}(a; n) = 1$, la funzione

$$f(x) := ax + b \tag{2.1}$$

è invertibile, e l'inversa è

$$f^{-1}(y) = a^{-1}(y - b) \ . \tag{2.2}$$

Chiaramente le operazioni di somma, prodotto ed inverso che compaiono nelle formule (2.1) e (2.2) vanno intese come operazioni in $\mathbb{Z}_n$, e non nel senso usuale.

Come si ottiene la formula (2.2)? Dato $y \in \mathbb{Z}_n$, $f^{-1}(y)$ è il numero x che soddisfa $y = f(x)$, ovvero $y = ax + b$; si tratta dunque di esplicitare x in funzione di y, e per farlo seguiamo il solito procedimento imparato a scuola: sottraendo b ad entrambe i termini dell'equazione otteniamo $y - b = ax$, e dividendo per a otteniamo $x = a^{-1}(y - b)$.[9]

2.6. Codici a chiave lineare

A questo punto viene spontaneo chiedersi se le funzioni *lineari*, cioè quelle del tipo $f(x) := ax + b$, possano essere utilizzate per costruire codici a chiave pubblica.

Tornando allo schema di illustrato in Sezione 1.6, B comunica ad A un numero n e la formula $f(x) = ax + b$, o più semplicemente i numeri a e b (ovviamente B avrà scelto a primo con n, in modo che la funzione f sia invertibile). A questo punto A ha tutto quel che gli serve per crittare i messaggi da spedire a B. Una terza persona C in grado di intercettare le comunicazioni tra

[9] Non avendo definito differenza e divisione in $\mathbb{Z}_n$, quello che veramente facciamo è sommare $-b$ e moltiplicare per a^{-1}.

A e B ha accesso sia al numero n che alla chiave di crittazione f, ma quello di cui avrebbe bisogno per poter decrittare il messaggio è la funzione f^{-1}, ovvero, ricordando la formula (2.2), del numero a^{-1}. Numero che B si è ben guardato dal trasmettere ad A!

Chiaramente C può trovare l'inverso di a per tentativi, cioè calcolando multipli di a fino a trovare quello congruo a 1 modulo n. Se è molto fortunato gli va bene al primo tentativo e se la cava con una sola moltiplicazione, ma se è molto sfortunato potrebbe doverne fare anche $n-1$. In media, avrà bisogno di $n/2$ moltiplicazioni prima di trovare l'inverso cercato. Viceversa, per crittare il messaggio da spedire, A deve calcolare $y_i = f(x_i)$ per ciascuna unità x_i del messaggio, e quindi deve fare un numero di moltiplicazioni e di somme pari al numero k di unità del messaggio. Ma se n è enormemente più grande di k, è plausibile che il compito di A sia realizzabile con l'aiuto di un qualunque personal computer nel giro di pochi attimi, mentre quello di C sia proibitivo anche per i computer più potenti. Se così fosse, avremmo ottenuto un buon codice a chiave pubblica!

A questo punto ci si rende conto che il tempo necessario per trovare l'inverso di a in $\mathbb{Z}_n$ dipende dalla procedura, o algoritmo, che utilizziamo: forse calcolare tutti i multipli di a non è il modo più veloce per trovarne l'inverso ... Ma che significa poi "più veloce"? Per poter procedere nella discussione, diventa inevitabile chiedersi di quanto tempo ha bisogno un computer per eseguire un certo algoritmo, che sia quello per sommare due numeri interi o quello per trovare l'inverso. Cercheremo di dare una risposta a questa domanda nel prossimo capitolo.

Esercizi

2.7. – Determinare l'inverso di 4 in $\mathbb{Z}_9$.

2.8. – Compilare la tabella (1.1) per la funzione $f(x) := 4x + 1$ su $\mathbb{Z}_9$. Costruire "a mano" la tabella dell'inversa, e verificare che corrisponde alla funzione f data dalla formula (2.2).

2.9. – Compilare la tabella (1.1) per la funzione $f(x) := 4x + 1$ su $\mathbb{Z}_8$. Verificare che in questo caso f non è invertibile.

2.10. – Se a non è primo con n allora non ammette inverso in $\mathbb{Z}_n$, e dunque la formula (2.2) per l'inversa della funzione $f(x) := ax+b$ non ha senso. Questo significa necessariamente che f non è invertibile? La risposta è sì; provare a dimostrarlo.

2.11. – Dimostrare che $f(x) := x^2$ non è invertibile in $\mathbb{Z}_n$ per alcun $n > 2$. Suggerimento: calcolare $f(1)$ e $f(n-1)$.

2.12. – Compilare la tabella (1.1) per la funzione $f(x) := x^3$ su $\mathbb{Z}_5$. Verificare che f è iniettiva e scrivere la tabella della funzione inversa.

2.13. – Verificare $f(x) := x^3$ non è iniettiva in $\mathbb{Z}_7$.

2.14. – Per quali numeri primi p la funzione $f(x) := x^3$ è invertibile in $\mathbb{Z}_p$? verificare uno ad uno i casi $p = 3, 5, 7, 11, 13$ e formulare una congettura per p generico.

2.15. – Applicare la procedura di crittazione descritta in Sezione 1.4 al messaggio `questo␣e␣un␣esercizio␣facile` con $n = 22$ ed $f(x) := 3x - 10$. Si deve quindi tradurre il messaggio in una sequenza di numeri in $\mathbb{Z}_{22}$ attribuendo ad ogni lettera il numero corrispondente alla posizione nell'ordine alfabetico e allo spazio vuoto ␣ il numero 0 (cfr. esercizio 1.15), e poi calcolare i valori di f corrispondenti. Se non vado errato il messaggio crittato dovrebbe essere

```
13,3,5,19,0,7,12,5,12,3,4,12,5,19,5,
16,21,17,9,17,7,12,8,15,21,17,20,5.
```

3. Valutare la velocità di un algoritmo

Come abbiamo detto, non si può discutere di codici a chiave pubblica senza entrare nel merito degli algoritmi utilizzati per le operazioni di crittazione e decrittazione e del tempo necessario per eseguirli.

Dato un certo algoritmo matematico, cercheremo quindi di determinare il numero di *operazioni elementari* di cui questo si compone, dove per operazioni elementari intenderemo le moltiplicazioni ed i prodotti di numeri di una cifra. Un po' impropriamente, chiameremo questo numero *costo computazionale* dell'algoritmo.

Il costo computazionale è un parametro significativo perché è indipendente dal mezzo (computer) utilizzato per eseguire l'algoritmo. Inoltre, possiamo calcolare in prima approssimazione il tempo necessario ad un determinato computer per eseguire questo algoritmo moltiplicandone il costo computazionale per il tempo medio necessario ad eseguire un'operazione elementare.[10] In particolare il rapporto tra i tempi necessari ad eseguire due diversi algoritmi è sostanzialmente uguale al rapporto dei costi computazionali, e quindi indipendente dal computer utilizzato.

Prima di proseguire, è bene ribadire un fatto essenziale. Per ottenere un certo risultato esistono in generale diverse procedure: ad esempio, se dobbiamo moltiplicare un numero intero per 4 possiamo farlo nel solito modo, oppure sommarlo a se stesso quattro volte, oppure ancora raddoppiarlo due volte. Il risultato è sempre lo stesso ma a seconda della procedura cambia il numero di

[10] Si tratta ovviamente di una drastica semplificazione, e ci sarebbe molto da obiettare: quelle che noi chiamiamo operazioni elementari non sono affatto tali per il processore di un computer; il processore non esegue una sola operazione elementare per volta, ma molte contemporaneamente; noi non teniamo conto di alcune operazioni, come lo spostamento di un dato da un registro di memoria ad un altro; infine un computer non opera in base decimale, a differenza di quanto faremo noi. Tuttavia il livello di imprecisione delle stime che daremo in seguito è tale da rendere superflua la discussione di questi dettagli.

operazioni elementari necessarie. Quando in seguito parleremo di costo computazionale di una certa operazione sottointenderemo sempre di starla facendo secondo un certo algoritmo, e nel caso che non sia ovvio quale, lo specificheremo.

3.1. Notazione per le stime

Supponiamo di voler determinare quante operazioni elementari sono richieste per compiere una certa operazione aritmetica (ad esempio, l'elevamento a potenza) su un numero intero x di h cifre. Di solito è praticamente impossibile calcolarne il numero esatto, ma può essere relativamente facile *stimarlo per eccesso* in termini di h: ad esempio, non è difficile vedere che per elevare x al quadrato bastano $6h^2$ operazioni elementari e che per moltiplicare x per 124 bastano $10h$ operazioni. Se, come nel nostro caso, si vuole confrontare il costo computazionale delle due operazioni quando h è molto grande, è chiaro che le parti rilevanti di queste stime non sono tanto la costanti 6 e 10 quanto gli esponenti 2 ed 1, che ci dicono che per numeri grandi la prima operazione è quella che richiede più tempo (anche se per numeri piccoli è vero il contrario).

Per sottolineare questo fatto diciamo che il costo computazionale della prima operazione è *dell'ordine di* h^2 e quello della la seconda è *dell'ordine di* h. Per indicare le grandezze dell'ordine di h^2 useremo genericamente il simbolo $O(h^2)$ (leggi: "o grande di acca al quadrato"). Più in generale, diciamo che il costo di un algoritmo dipendente da una certa grandezza h è dell'ordine di h^a, ovvero è $O(h^a)$, quando è inferiore a ch^a per un'opportuna scelta della costante c e dell'esponente a.

Osservazioni – 1. La notazione degli "o grandi" è comoda per molti motivi. Innanzitutto permette trascurare la costante c e di combinare agevolmente diverse stime: dati due algoritmi con costo computazionale $O(h^a)$, eseguirli entrambi richiederà un costo computazionale dello stesso ordine;[11] eseguire h volte un algoritmo con costo computazionale $O(h^a)$ risulterà in un costo computazionale $O(h^{a+1})$.[12]

2. Stando alla definizione, dire che il costo computazionale di un certo algoritmo è dell'ordine di h^a significa solo che il numero di operazioni elementari di cui si compone è inferiore a ch^a per una qualche costante c. Ciò non toglie che questo numero possa anche essere inferiore a $\tilde{c}h^b$ per qualche esponente b inferiore ad a. In altre parole, dire che una grandezza è dell'ordine di h^a non esclude che sia anche dell'ordine di h^b per qualche $b < a$. Diciamo sin da ora che, anche se non ci periteremo di dimostrarlo, tutte le stime che daremo in seguito sono ottimali, nel senso che gli esponenti non possono essere ulteriormente abbassati a meno di non utilizzare algoritmi migliori.

[11] Questa osservazione è sintetizzata dalla regola $O(h^a) + O(h^a) = O(h^a)$. Più in generale si ha che $O(h^a) + O(h^b) = O(h^a)$ se $b \leq a$ (cfr. esercizio 3.6).

[12] Questa osservazione è sintetizzata dalla regola $h \cdot O(h^a) = O(h^{a+1})$. Più in generale si ha che $O(h^b) \cdot O(h^a) = O(h^{a+b})$ (cfr. esercizio 3.6).

3. Una stima espressa in termini di "o grandi" ha purtroppo un valore solamente qualitativo: sapere che un algoritmo richiede un numero di passi inferiore a ch^a senza conoscere il valore della costante c non permette di fare alcuna previsione. Per ragioni di semplicità espositiva conviene nondimeno utilizzare questa notazione. Il lettore paziente (ma davvero paziente) può in teoria ricostruire le costanti mancanti a partire dalla descrizione degli algoritmi usati, soprattutto se non cerca le migliori costanti possibili.

3.2. Stima per il prodotto

Consideriamo due numeri interi x ed y di k ed h cifre rispettivamente, con $k \geq h$. Se moltiplichiamo x ed y secondo il metodo imparato a scuola, il numero delle operazioni coinvolte è dell'ordine di kh. Consideriamo infatti lo svolgimento della moltiplicazione di $x = 1435$ per $y = 276$:

$$\begin{array}{r} 1435 \\ 276 \\ \hline 8610 \\ 10045 \\ 2870 \\ \hline 396060 \end{array}$$

La terza riga è stata ottenute moltiplicando 1435 per l'ultima cifra di 276, e cioè 6, e questa operazione comporta moltiplicare 6 per ciascuna delle cifre del di 1435 e tenere conti di alcuni riporti. La quarta e la quinta riga sono state ottenute moltiplicando 1435 per 7 e per 2, rispettivamente. Infine per ottenere il risultato nella sesta riga abbiamo sommato le righe dalla terza alla quinta.

In generale si moltiplica x per ognuna delle h cifre di y ottenendo i numeri $z_1, \dots, z_h$, e ciascuna operazione richiede k moltiplicazioni elementari più un numero non superiore a k di riporti, per un totale di $h(k + k)$ operazioni elementari. I numeri $z_1, \dots, z_h$ contano al più $k + 1$ cifre, e vanno sommati opportunamente, per un totale di $(h - 1)(k + 1)$ somme elementari ed un numero paragonabile di riporti. Il numero delle operazioni elementari richiesto è quindi

$$2hk + 2(h - 1)(k + 1) \leq 2hk + 4hk = 6hk = O(hk) \ .$$

Siccome il numero di cifre di un numero corrisponde all'incirca al suo logaritmo in base 10 (cfr. esercizio 1.10) possiamo anche dire che il costo computazionale della moltiplicazione di due numeri x ed y, eseguita secondo l'algoritmo classico, è $O(\log x \log y)$.

Osservazione – L'algoritmo per moltiplicare due numeri interi considerato nel paragrafo precedente non è quello con costo computazionale minore. Esistono algoritmi per moltiplicare due numeri di k cifre con costo $O(k \log k \log\log k)$ invece di $O(k^2)$. Si noti che già per k dell'ordine di 10^3 la differenza tra k^2 e $k \log k \log\log k$ è notevole.

3.3. Stime per altre operazioni

Sommare x ed y secondo l'algoritmo imparato a scuola comporta h somme elementari ed al più h riporti. Dunque la somma ha un costo computazionale inferiore a $2h = O(h)$, ovvero dell'ordine del logaritmo dell'addendo più piccolo.

Allo stesso modo si può far vedere che la divisione con resto di x per y secondo l'algoritmo imparato a scuola ha costo $O(kh)$, ovvero $O(\log x \log y)$ (cfr. esercizio 3.7).

3.4. Stime per le operazioni in $\mathbb{Z}_n$

Per moltiplicare due numeri x e y in $\mathbb{Z}_n$, possiamo calcolare il prodotto di x per y nel modo usuale, e poi ridurlo ad un numero in $\mathbb{Z}_n$, vale a dire dividerlo per n e prendere il resto di tale operazione (cfr. Sezione 2.1). Stimando il numero di cifre di x ed y con $\log n$, la prima operazione ha costo $O(\log^2 n)$ (cfr. Sezione 3.2) come pure la seconda (cfr. Sezione 3.3). Dunque la moltiplicazione di due numeri in $\mathbb{Z}_n$ secondo questo algoritmo ha costo $O(\log^2 n)$.

Allo stesso modo si vede che la somma di due numeri in $\mathbb{Z}_n$ ha costo $O(\log n)$.

Quanto tempo serve invece per trovare l'inverso di un numero a in $\mathbb{Z}_n$? Se adottiamo l'algoritmo più semplice, ovvero calcoliamo tutti i multipli di a fino a che non otteniamo 1, allora il costo computazionale è $O(n \log^2 n)$. Infatti ogni moltiplicazione ha costo $O(\log^2 n)$ e in media il numero di tentativi da fare sarà dell'ordine di n.

3.5. Ancora sui codici a chiave lineare

Supponiamo ora di voler implementare il codice descritto in Sezione 2.6, cioè un codice con chiave pubblica $f(x) := ax + b$ e chiave privata $f^{-1}(x) := a^{-1}(x-b)$, Per capire se questo può essere un buon codice a chiave pubblica dobbiamo confrontare il costo computazionale della crittazione e decrittazione di un messaggio (cioè le operazioni svolte da A e B) con quello della ricostruzione della chiave privata a partire dalla chiave pubblica (l'operazione preliminare che C deve fare per poter decrittare i messaggi).

Calcolare il valore di $f(x)$ per un dato $x \in \mathbb{Z}_n$ richiede una moltiplicazione ed una somma e per quanto visto nel paragrafo precedente ha costo computazionale

$$O(\log^2 n) + O(\log n) = O(\log^2 n) \ .$$

Un messaggio di lunghezza h scritto nel solito alfabeto di 22 caratteri viene tradotto in una sequenza di numeri in $\mathbb{Z}_n$ di lunghezza $k = O(h/\log n)$ (vedi esercizio 1.17) e l'operazione di crittazione consiste nel calcolare il valore di f per ciascuno di questi numeri. Pertanto il costo della crittazione del messaggio è

$$k \cdot O(\log^2 n) = O(h/\log n) \cdot O(\log^2 n) = O(h \log n) \ . \tag{3.1}$$

Una volta nota la chiave privata, e cioè il numero a^{-1}, il costo dell'operazione di decrittazione è ovviamente dello stesso ordine.

Ricavare la chiave privata a partire dalla chiave pubblica significa invece determinare il numero a^{-1} a partire da a, e come abbiamo visto nel paragrafo precedente il costo è

$$O(n \log^2 n) \ . \tag{3.2}$$

Se le cose stessero davvero così, allora avremmo un ottimo codice a chiave pubblica: prendendo un messaggio di lunghezza $h = 1000$ ed $n = 10^{50}$, la crittazione e decrittazione del messaggio richiederebbero un numero di operazioni elementari dell'ordine di $h \log n = 10^4$, cioè pochi istanti di tempo macchina di un qualunque personal computer, mentre la ricostruzione della chiave privata a partire dal quella pubblica richiederebbe un numero di operazioni elementari dell'ordine di 10^{53}, e quindi al di fuori della portata di qualunque computer: anche supponendo che la frequenza del processore sia di 10 GigaHertz, ovvero di 10^{10} cicli al secondo, il tempo necessario sarebbe 10^{43} secondi, cioè 10^{26} miliardi di anni.[13]

Purtroppo le cose non stanno affatto così, perché come vedremo nel prossimo capitolo esiste un algoritmo, basato sull'algoritmo di Euclide per il calcolo del massimo comun divisore, che permette di trovare l'inverso di un numero in $\mathbb{Z}_n$ con costo computazionale $O(\log^3 n)$: questo significa che per $n = 10^{50}$ la ricostruzione della chiave privata richiederebbe circa 10^5 operazioni elementari, cioè appena di più di quelle necessarie per le operazioni di crittazione e decrittazione. Dunque le funzioni del tipo $f(x) := ax + b$ non possono essere utilizzate come chiavi in un codice a chiave pubblica.

Osservazione – Per ragioni di semplicità espositiva, le stime nel paragrafo precedente sono state fatte in modo volutamente approssimativo. Anche se sappiamo che le chiavi lineari non possono essere utilizzate per un codice a chiave pubblica, è bene cercare di capire quali errori abbiamo commesso ed in che modo hanno alterato il risultato finale.

1. Nel calcolo del costo computazionale delle operazioni di crittazione e decrittazione non abbiamo tenuto conto del tempo necessario ad A e B per trasformare il messaggio in sequenze di numeri in $\mathbb{Z}_n$.

2. Al momento di far vedere che il costo computazionale delle operazioni di crittazione e decrittazione è trascurabile rispetto a quello del calcolo dell'inverso di a, abbiamo supposto che le stime (3.1) e (3.2) si riducessero a $h \log n$ e $n \log^2 n$, trascurando quindi le costanti moltiplicative nascoste dalla notazione degli "o grandi" (cfr. Sezione 3.1 e successive osservazioni). Un esame più accurato—che omettiamo—mostrerebbe che correggere questo errore non altererebbe il stima finale in modo significativo.

3. Un'omissione ben più rilevante è invece la seguente: per utilizzare questo codice B deve essere in grado di generare coppie di numeri n ed a in modo da conoscere l'inverso a^{-1} in $\mathbb{Z}_n$. Si noti che tale compito non è a prima vista

[13] Si noti che nel contesto di tali numeri incrementare anche di un fattore 1000 la velocità del computer, ovvero usare 1000 processori che lavorano in parallelo, servirebbe a ben poco.

molto diverso da quello di calcolare l'inverso di a, che è appunto ciò che deve fare C per ricavare la chiave privata.

Esercizi

3.6. – Siano f e g funzioni da $\mathbb{N}$ in $\mathbb{N}$, con $g(h) \geq 1$ per $h \geq 1$. Scriviamo $f = O(g)$ se esiste una costante c tale che $f(h) \leq c\,g(h)$ per ogni $h \geq 1$. Dimostrare che:
(a) $h^b = O(h^a)$ se e solo se $b \leq a$;
(b) se $f_1 = O(g)$ e $f_2 = O(g)$ allora $f_1 + f_2 = O(g)$;
(c) se $f_1 = O(g_1)$ e $f_2 = O(g_2)$ allora $f_1 \cdot f_2 = O(g_1 \cdot g_2)$.

3.7. – Siano x ed y numeri interi positivi rispettivamente di k e h cifre, con $k \geq h$. Far vedere che il costo computazionale della divisione con resto di x per y secondo l'algoritmo imparato a scuola è $O(h(k-h+1))$.

3.8. – Si consideri l'algoritmo proposto in Sezione 1.14 per trasformare un messaggio scritto nel solito alfabeto di 22 lettere in una sequenza di numeri in $\mathbb{Z}_n$. Stimarne il costo computazionale in termini di n e della lunghezza h del messaggio. Stimare poi il costo dell'algoritmo semplificato proposto nell'esercizio 1.16 per $n = 22^m$.

3.9. – Abbiamo visto che la moltiplicazione di due numeri interi di k cifre secondo l'algoritmo usuale ha costo computazionale $O(k^2)$. Verificare che questa stima è ottimale, ovvero che l'esponente 2 non può essere sostituito con un numero inferiore (a meno ovviamente di non utilizzare un algoritmo diverso—cfr. Sezione 3.2 e successiva osservazione).

4. L'algoritmo di Euclide

Il metodo che tutti conosciamo per determinare il massimo comun divisore di due numeri si basa sulla scomposizione in fattori primi, operazione notoriamente lunga e complicata. L'algoritmo di Euclide permette di calcolare il massimo comun divisore senza ricorrere alla fattorizzazione; ma quello che più ci interessa è un "sottoprodotto" di questo algoritmo che permette di trovare l'inverso di un numero in $\mathbb{Z}_n$ con costo $O(\log^3 n)$ invece di $O(n \log^2 n)$.

Tutto si basa su una semplice osservazione:

LEMMA 4.1 – *Siano a e b numeri interi positivi con $a > b > 1$, e sia c il resto della divisione di a per b. Allora* $\text{MCD}(a; b) = \text{MCD}(b; c)$.

DIMOSTRAZIONE – Sia d il quoziente della divisione di a per b; allora

$$a = bd + c\ . \tag{4.1}$$

Ne consegue che ogni numero che divide sia b che c deve dividere anche a. In particolare ogni divisore comune di b e c è anche un divisore comune di a e b. Riscrivendo la (4.1) come

$$c = a - bd$$

otteniamo invece che ogni divisore comune di a e b è anche un divisore comune di b e c. □

4.2. L'algoritmo di Euclide

Siano a_1 ed a_2 numeri interi positivi con $a_1 > a_2 > 1$. Per via del Lemma 4.1,

$$\text{MCD}(a_1; a_2) = \text{MCD}(a_2; a_3)$$

dove a_3 è il resto della divisione di a_1 per a_2. Per la stessa ragione,

$$\text{MCD}(a_2; a_3) = \text{MCD}(a_3; a_4)$$

dove a_4 è il resto della divisione di a_2 per a_3. Definiamo $a_5, a_6, \dots$ a questo modo fino a quando a_k è zero[14] ovvero a_{k-1} divide esattamente a_{k-2}. Allora

$$\text{MCD}(a_1; a_2) = \text{MCD}(a_{k-2}; a_{k-1}) = a_{k-1} \ .$$

Mettiamo in pratica questo algoritmo per calcolare il massimo comun divisore di $a_1 := 1547$ ed $a_2 := 560$:

$$\begin{aligned} 1547 : 560 = 2 \quad &\text{con resto} \quad a_3 = 427; \\ 560 : 427 = 1 \quad &\text{con resto} \quad a_4 = 133; \\ 427 : 133 = 3 \quad &\text{con resto} \quad a_5 = 28; \\ 133 : 28 = 4 \quad &\text{con resto} \quad a_6 = 21; \\ 28 : 21 = 1 \quad &\text{con resto} \quad a_7 = 7; \\ 21 : 7 = 3 \quad &\text{con resto} \quad a_8 = 0. \end{aligned}$$

Quindi $\text{MCD}(1547; 560) = 7$ (ottenuto in 6 passi).

4.3. Stima del numero di passi

Prendiamo $a_1, a_2, \dots, a_k$ come nel paragrafo precedente. Il numero di passi, ovvero di divisioni con resto, fatti per arrivare alla conclusione è $k - 2$. Se indichiamo con q_i il risultato della divisione di a_i per a_{i+1}, allora $a_i = q_i a_{i+1} + a_{i+2}$, e siccome $a_{i+1} > a_{i+2}$ (cfr. Nota 14)

$$a_i = q_i a_{i+1} + a_{i+2} \geq a_{i+1} + a_{i+2} > 2a_{i+2} \ .$$

Dunque ogni due passi il valore di a_i risulta almeno dimezzato. Se ne deduce (cfr. esercizio 4.7) la seguente stima sul numero di passi dell'algoritmo di Euclide:

$$2\frac{\log a_1}{\log 2} + 1 \geq k - 2 \ .$$

[14] Siccome il resto di una divisione è sempre strettamente minore del divisore, i numeri $a_1, a_2, a_3, \dots$ formano una sequenza strettamente decrescente di numeri interi positivi, che deve prima o poi raggiungere il valore 0.

4.4. Una conseguenza importante

Usando l'algoritmo di Euclide possiamo scrivere il massimo comun divisore di a_1 ed a_2 come somma di due multipli (positivi o negativi) di a_1 ed a_2.

Prendiamo $a_1, a_2, a_3, \ldots, a_k$ come in Sezione 4.3, ed indichiamo con q_i il risultato della divisione di a_i per a_{i+1}. Allora

$$a_{i+2} = a_i - q_i a_{i+1} \quad \text{per } i = 1, 2, \ldots, k-2. \tag{4.2}$$

Sappiamo che $\text{MCD}(a_1; a_2) = a_{k-1}$, e ponendo $i := k-3$ nella (4.2) otteniamo la formula

$$\text{MCD}(a_1; a_2) = a_{k-1} = a_{k-3} - q_{k-3} a_{k-2} \ , \tag{4.3}$$

che esprime $\text{MCD}(a_1; a_2)$ come somma di multipli interi di a_{k-2} ed a_{k-3}. Ponendo $i := k-4$ nella (4.2) otteniamo a_{k-2} come somma di multipli interi di a_{k-4} ed a_{k-3}, e sostituendo questa espressione nella (4.3) otteniamo $\text{MCD}(a_1; a_2)$ come somma di multipli interi di a_{k-3} ed a_{k-4}:

$$\begin{aligned} \text{MCD}(a_1; a_2) &= a_{k-3} - q_{k-3} a_{k-2} \\ &= a_{k-3} - q_{k-3}(a_{k-4} - q_{k-4} a_{k-3}) \\ &= (1 + q_{k-4}) a_{k-3} - q_{k-3} a_{k-4} \ . \end{aligned} \tag{4.4}$$

A questo punto è chiaro qual'è il passo seguente: si usa la (4.2) con $i := k-5$ per scrivere a_{k-3} come somma di multipli di a_{k-4} ed a_{k-5}, e si sostituisce l'espressione così trovata nell'ultima riga della (4.4), che diventa quindi una somma di multipli interi di a_{k-5} ed a_{k-4}. Si procede in questo modo fino ad ottenere $\text{MCD}(a_1; a_2)$ come somma di multipli di a_1 ed a_2.

Per capire, vediamo cosa succede nel caso $a_1 = 1547$ ed $a_2 = 560$. Innanzitutto, elenchiamo le formule (4.2) al variare di $i = 1, \ldots, 5$:

$$\begin{aligned} 427 &= 1547 - 2 \cdot 560, \\ 133 &= 560 - 1 \cdot 427, \\ 28 &= 427 - 3 \cdot 133, \\ 21 &= 133 - 4 \cdot 28, \\ 7 &= 28 - 1 \cdot 21. \end{aligned}$$

Quindi, procedendo come si è detto,

$$\begin{aligned} \text{MCD}(1547; 560) &= 7 \\ &= 28 - 1 \cdot 21 \\ &= 28 - 1 \cdot (133 - 4 \cdot 28) = -133 + 5 \cdot 28 \\ &= -133 + 5 \cdot (427 - 3 \cdot 133) = 5 \cdot 427 - 16 \cdot 133 \\ &= 5 \cdot 427 - 16 \cdot (560 - 1 \cdot 427) = -16 \cdot 560 + 21 \cdot 427 \\ &= -16 \cdot 560 + 21 \cdot (1547 - 2 \cdot 560) = 21 \cdot 1547 - 58 \cdot 560, \end{aligned}$$

e dunque $\text{MCD}(1547; 560) = 7 = 21 \cdot 1547 - 58 \cdot 560$.

4.5. Un nuovo algoritmo per il calcolo dell'inverso

Se $a \in \mathbb{Z}_n$ è primo con n, ovvero $\text{MCD}(a; n) = 1$, per quanto visto nel paragrafo precedente possiamo scrivere 1 come somma di multipli interi (positivi o negativi) di n ed a, ovvero trovare dei numeri $r, s \in \mathbb{Z}$ tali che

$$1 = r \cdot a + s \cdot n \ .$$

Ma questo significa che $1 \equiv r \cdot a \mod n$, e dunque

$$r \equiv a^{-1} \mod n.$$

Abbiamo così trovato una nuova dimostrazione del fatto che ogni numero a primo con n è invertibile in $\mathbb{Z}_n$. Inoltre il numero r può essere ottenuto tramite la procedura descritta in Sezione 4.4.

Vogliamo ora determinare il costo computazionale di questo algoritmo. Osserviamo che la prima fase, cioè il calcolo dei numeri a_i e q_i, corrisponde all'algoritmo di Euclide descritto in Sezione 4.2 e consta di $k-2$ divisioni con resto, ciascuna delle quali ha costo $O(\log^2 n)$ perché i numeri coinvolti sono inferiori ad n; inoltre $k-2$, è dell'ordine di $\log n$ (Osservazione 4.3) e dunque il costo complessivo di questa prima fase è $O(\log^3 n)$. La procedura utilizzata in Sezione 4.4 per scrivere il massimo comun divisore di due numeri come somma di multipli dei numeri stessi consta di $k-2$ passaggi, ciascuno dei quali contiene un paio di somme e prodotti, e dunque anch'essa ha un costo dell'ordine di $O(\log^3 n)$.

Quindi questo algoritmo per il calcolo dell'inverso di a ha costo computazionale $O(\log^3 n)$.

Esercizi

4.6. – Sia $a_1, a_2, \dots, a_k$ una sequenza di numeri interi strettamente positivi ed $r > 1$ un numero reale tale che $a_i \geq r a_{i+1}$ per $i = 1, 2, \dots, k-1$. Dimostrare che $a_1 \geq r^{k-1} a_k \geq r^k$ e dedurne la stima

$$\frac{\log a_1}{\log r} + 1 \geq k \ .$$

4.7. – Sia $a_1, a_2, \dots, a_k$ una sequenza di numeri interi strettamente positivi ed $r > 1$ un numero reale tale che $a_i \geq r a_{i+2}$ per $i = 1, 2, \dots, k-2$. Dimostrare che

$$2\frac{\log a_1}{\log r} + 2 \geq k \ .$$

4.8. – Calcolare il massimo comun divisore di 672330 e 49531.

4.9. – Calcolare l'inverso di 49531 modulo 672330.

5. Il piccolo teorema di Fermat ed il codice RSA

Abbiamo visto in Sezione 3.5 che le funzioni del tipo $f(x) := ax + b$ non sono utilizzabili per costruire codici a chiave pubblica. In questa lezione esamineremo un'altra classe di funzioni da $\mathbb{Z}_n$ in sé, e cioè quelle del tipo

$$f(x) := x^a \tag{5.1}$$

con a intero positivo (ovviamente l'elevamento a potenza va inteso come prodotto iterato di x per se stesso nel senso di $\mathbb{Z}_n$). Per poter utilizzare una di queste funzioni in un codice, dobbiamo innanzitutto capire per quali a ed n risulta invertibile. Una risposta parziale a questa domanda segue da un noto risultato di teoria dei numeri:

Teorema 5.1 (Piccolo teorema di Fermat) – *Se p è un numero primo, allora per ogni $x \in \mathbb{Z}_p$ con $x \neq 0$ si ha*

$$x^{p-1} \equiv 1 \mod p. \tag{5.2}$$

Dimostrazione – Abbiamo già osservato nella dimostrazione del Teorema 2.4 che l'insieme dei multipli di x

$$\{x, 2x, 3x, \ldots, (p-1)x\}\ , \tag{5.3}$$

inteso come sottoinsieme di $\mathbb{Z}_p$, contiene $p-1$ elementi distinti e tutti diversi da 0. Pertanto questo insieme deve coincidere con

$$\{1, 2, 3, \ldots, p-1\}\ . \tag{5.4}$$

Prendendo allora il prodotto dei numeri in (5.3) e quello dei numeri in (5.4) dobbiamo ottenere lo stesso numero, ovvero

$$x \cdot 2x \cdot 3x \cdots (p-1)x \equiv 1 \cdot 2 \cdot 3 \cdots (p-1) \mod p. \tag{5.5}$$

Semplificando i fattori $2, 3 \ldots, p-1$ otteniamo infine l'identità (5.2).[15] □

Corollario 5.2 – *Se p è un numero primo ed a è un numero intero tale che* $\text{MCD}(a; p-1) = 1$*, allora la funzione $f(x) := x^a$ è invertibile su $\mathbb{Z}_p$, e l'inversa è data da*

$$f^{-1}(y) = y^{\bar{a}}\ , \tag{5.6}$$

dove $\bar{a}$ è un qualsiasi intero positivo tale che $a\bar{a} \equiv 1 \mod p-1$.

[15] Semplificare i fattori $2, 3 \ldots, p-1$ significa moltiplicare entrambi i membri della (5.5) per gli inversi di questi numeri, che esistono perché sono tutti primi con p. È proprio qui che si usa il fatto che p è un numero primo e non un intero qualunque.

Osservazione – Il numero $\bar{a}$ deve essere congruente all'inverso di a in $\mathbb{Z}_{p-1}$, che esiste per via dell'ipotesi $\text{MCD}(a; p-1) = 1$. Ovviamente i numeri $\bar{a}$ ammissibili sono infiniti, ma le funzioni potenza ad essi associate sono tutte uguali (questo fatto non è a prima vista evidente).

Dimostrazione – Per dimostrare che la funzione data dalla formula (5.6) è proprio l'inversa di f basta verificare che $f^{-1}(f(x)) = x$ per ogni $x \in \mathbb{Z}_p$, ovvero

$$(x^a)^{\bar{a}} \equiv x \mod p. \tag{5.7}$$

Questa identità è chiaramente vera per $x = 0$; resta da dimostrarla nel caso $x \neq 0$. Siccome $a\bar{a} \equiv 1 \mod p-1$, esiste b intero tale che $a\bar{a} = 1 + b(p-1)$; dunque

$$(x^a)^{\bar{a}} = x^{a\bar{a}} = x^{1+b(p-1)} = x \cdot (x^{p-1})^b \ ,$$

e per ottenere la (5.7) ci basta ricordare che $x^{p-1} \equiv 1 \mod p$ per via del piccolo teorema di Fermat. □

Questo corollario dà un criterio soddisfacente per l'invertibilità delle funzioni del tipo (5.1) su $\mathbb{Z}_p$ con p primo. Tuttavia per le applicazioni che vedremo in seguito ci interessa sapere cosa succede in $\mathbb{Z}_n$ con n numero non primo, e ci servirà la seguente generalizzazione del Corollario 5.2:

Proposizione 5.3 – *Se n è il prodotto di due primi distinti p e q, ed a è un numero intero tale che*

$$\text{MCD}(a; (p-1)(q-1)) = 1 \ ,$$

allora la funzione $f(x) := x^a$ è invertibile su $\mathbb{Z}_n$, e l'inversa è data da

$$f^{-1}(y) = y^{\bar{a}} \ , \tag{5.8}$$

dove $\bar{a}$ è un qualsiasi intero positivo tale che $a\bar{a} \equiv 1 \mod (p-1)(q-1)$.

Dimostrazione – Dobbiamo far vedere che per ogni intero x si ha

$$x^{a\bar{a}} \equiv x \mod n. \tag{5.9}$$

L'ipotesi $a\bar{a} \equiv 1 \mod (p-1)(q-1)$ implica che $(p-1)(q-1)$ divide $a\bar{a} - 1$ e quindi anche $p-1$ divide $a\bar{a} - 1$, ovvero

$$a\bar{a} \equiv 1 \mod p-1.$$

Dalla formula (5.7) otteniamo allora che $x^{a\bar{a}} \equiv x \mod p$ per ogni intero x, e dunque p divide $x^{a\bar{a}} - x$. Lo stesso ragionamento si applica a q, e dunque anche q divide $x^{a\bar{a}} - x$. Siccome p e q sono primi distinti, ne segue che pq divide $x^{a\bar{a}} - x$, ovvero la (5.9). □

5.4 Stima per l'elevamento a potenza

Per poter discutere l'uso delle funzioni di tipo $f(x) := x^a$ in un codice, dobbiamo stimare il costo computazionale del calcolo di x^a in $\mathbb{Z}_n$. Supporremo sempre $a \leq n$, che è l'unico caso che ci interessi.

Per questa operazione la scelta dell'algoritmo si fa sentire: essendo il costo di una moltiplicazione $O(\log^2 n)$, moltiplicare x per se stesso a volte comporterebbe un costo $O(a \log^2 n) = O(n \log^2 n)$.

Tuttavia c'è un modo migliore per calcolare la potenza di un numero: se ad esempio a è una potenza di 2, cioè $a = 2^h$, allora calcolare x^a significa elevare x al quadrato h volte; essendo il costo di ciascuna operazione di elevamento al quadrato $O(\log^2 n)$ ed essendo h dell'ordine di $\log a$,[16] il costo complessivo di questo algoritmo è

$$O(\log a \log^2 n) = O(\log^3 n) \ .$$

Modificando questa idea si può trovare un algoritmo per il calcolo di x^a in $\mathbb{Z}_n$ con costo $O(\log^3 n)$ anche quando a non è una potenza di 2 (vedi Sezione 5.13).

5.5. Il codice a chiave pubblica RSA

Uno dei codici a chiave pubblica usati più comunemente è quello proposto da R. Rivest, A. Shamir e L. Adleman nel 1977, e noto come RSA dalle iniziali dei loro cognomi. Questo codice usa come chiavi pubbliche proprio funzioni del tipo $f(x) := x^a$.

Cercheremo ora di capire perché tali funzioni diano luogo ad un buon codice a chiave pubblica. Per garantire che f sia invertibile, supponiamo che valgano le ipotesi della Proposizione 5.3:

(i) n è il prodotto di primi distinti p e q,
(ii) a è primo con $(p-1)(q-1)$;
(iii) $a \leq n$.

Come al solito, dobbiamo confrontare il costo computazionale della crittazione e decrittazione di un messaggio (cioè le operazioni svolte da A e B) con quello della ricostruzione della chiave privata f^{-1} a partire dalla chiave pubblica f (l'operazione preliminare che C deve fare per poter decrittare i messaggi).

Per quanto detto in Sezione 5.4 il costo computazionale del calcolo di $f(x)$ è $O(\log^3 n)$. Un messaggio di lunghezza h viene trasformato in una sequenza di numeri x_i in $\mathbb{Z}_n$ di lunghezza $O(h/\log n)$, e la crittazione di questa sequenza, cioè il calcolo di $y_i = f(x_i)$ per ogni i, ha un costo pari a

$$O(h/\log n) \cdot O(\log^3 n) = O(h \log^2 n) \ . \tag{5.10}$$

Visto che f^{-1} è una funzione dello stesso tipo di f, il costo dell'operazione di decrittazione è dello stesso ordine di grandezza.

---

[16] Per la precisione h è il logaritmo in base 2 di a, che differisce dal logaritmo in base 10 solo per un fattore costante.

Passiamo ora alla ricostruzione della chiave privata: per via della Proposizione 5.3, trovare f^{-1} significa trovare l'inverso di a in $\mathbb{Z}_{(p-1)(q-1)}$, ed abbiamo già visto in Sezione 4.5 che il costo di questa operazione è $O(\log^3 n)$ (si ricordi che $(p-1)(q-1) \leq n$). Tuttavia per poter applicare l'algoritmo descritto in Sezione 4.5 e trovare l'inverso di a, C deve conoscere il numero $(p-1)(q-1)$.

Questo è il punto: C conosce a ed n, che sono parte della chiave pubblica, ma non i fattori p e q, che non sono necessari all'operazione di crittazione e che quindi B si è ben guardato dal divulgare. Ovviamente C può calcolare p e q fattorizzando n, ma questa è notoriamente un operazione con costi computazionali enormi, in particolare se n è il prodotto di due soli fattori primi di grandezza comparabile.

Se prendiamo n dell'ordine di 10^{300}, allora le operazioni di crittazione e decrittazione di un messaggio di $h = 1000$ caratteri richiedono un numero di operazioni elementari dell'ordine di $h\log^2 n = 10^8$, che un processore da 1 GigaHertz può eseguire in meno di un secondo. Viceversa fattorizzare un numero di 300 cifre decimali è a tutt'oggi oltre i limiti del possibile.[17]

Osservazioni – 1. La sicurezza dell'RSA risiede proprio nel fatto che la ricostruzione della chiave pubblica a partire dalla chiave privata equivale a fattorizzare n, operazione per cui non si conoscono algoritmi sufficientemente veloci: è ragionevole supporre che in tanti secoli che i matematici studiano i numeri primi, se c'era un modo veloce di fattorizzare un numero qualcuno lo avrebbe trovato. Per quanto ragionevole, si tratta però solo di una supposizione, e nessuno ha mai dimostrato che tali algoritmi non possano esistere.[18]

2. Nel calcolare il costo delle operazioni di crittazione e decrittazione abbiamo nuovamente trascurato di considerare la conversione del messaggio in sequenza di numeri in $\mathbb{Z}_n$ e viceversa. Includerle non altererebbe in modo significativo la stima (5.10).

3. Un altro passo della procedura di crittazione e decrittazione che abbiamo omesso di considerare, assai più rilevante del precedente, è questo: per implementare il codice RSA, è necessario che B sia in grado di trovare numeri primi p e q piuttosto grandi in tempi molto più brevi di quelli necessari a C per fattorizzare di n. Trovare numeri primi grandi in tempi relativamente brevi è fattibile,

[17] Alla metà degli anni settanta fattorizzare un numero di 40-50 cifre decimali era considerato molto difficile, all'inizio degli anni ottanta si fattorizzavano di routine numeri dalle 60 alle 90 cifre, e verso la fine degli anni novanta si è arrivati a numeri di oltre 150 cifre, cosa che però ha richiesto il lavoro congiunto di diverse decine di computer per quasi un centinaio di ore. Questi progressi, per quanto a prima vista impressionanti, sono essenzialmente dovuti all'incremento della velocità dei computer e non all'invenzione di nuovi algoritmi di fattorizzazione, e non pongono seri problemi alla sicurezza dell'RSA.

[18] Si tratterebbe di dimostrare che il costo computazionale di *ogni* algoritmo di fattorizzazione non può essere più piccolo di tanto. Risultati di questo tipo sono in generale molto più difficili da ottenere che le stime per eccesso sul costo di un singolo algoritmo; ad esempio, non si è ancora riusciti a dimostrare che non ci sono algoritmi per la moltiplicazione di due numeri di k cifre con costo $O(k)$.

ma la spiegazione va oltre l'orizzonte di queste note. Deve essere tuttavia ben chiaro che si tratta di un punto essenziale—in effetti, tra le operazioni che B deve compiere, questa è quella dal costo computazionale maggiore.

4. Se è vero che non si conoscono (per ora) algoritmi di fattorizzazione efficienti per ogni numero n, tuttavia ne esistono alcuni che sono piuttosto veloci quando n ha una forma particolare. Ad esempio, è relativamente facile fattorizzare numeri della forma $n = pq$ con p e q primi molto vicini tra di loro. Dunque l'affidabilità del codice dipende in modo essenziale da come vengono scelti i numeri p e q. Ma questa è un'altra storia.

5. Abbiamo detto che per trovare il numero $\bar{a}$ e ricostruire la chiave privata, C ha bisogno di conoscere il numero $m := (p-1)(q-1)$, e quindi la fattorizzazione di n. Ma le cose stanno veramente così? oppure esiste una scorciatoia che permette di trovare m senza dover fattorizzare n? (così come l'algoritmo di Euclide permette di trovare il massimo comun divisore di due numeri senza doverli fattorizzare). La risposta è che tale scorciatoia non esiste, nel senso che conoscere sia n che m è *equivalente* a conoscere i fattori primi p e q. Infatti le equazioni $n = pq$ e $m = (p-1)(q-1)$ implicano

$$\begin{cases} pq = n \ , \\ p + q = n - m + 1 \ , \end{cases}$$

quindi p e q sono le soluzioni dell'equazione di secondo grado

$$x^2 - (n - m + 1)x + n = 0$$

e possono essere facilmente calcolati a partire da n ed m.

5. Finora abbiamo dato per scontato il fatto che per ricostruire la chiave privata f^{-1} sia necessario calcolare l'inverso di a modulo $(p-1)(q-1)$, e che in particolare sia necessario conoscere $(p-1)(q-1)$. Questo non è del tutto vero: come osservato in Sezione 5.12 basta calcolare l'inverso di a modulo m con m un qualunque multiplo comune di $p-1$ e $q-1$, e quindi il problema si riduce a trovare un tale multiplo. Si può però dimostrare che anche questa operazione è sostanzialmente equivalente a fattorizzare n.

Complementi ed esercizi

5.6. Una generalizzazione del piccolo teorema di Fermat – Dato n intero positivo qualunque, sia $\mathscr{I}_n$ l'insieme dei numeri invertibili in $\mathbb{Z}_n$, vale a dire

$$\mathscr{I}_n := \{k \in \mathbb{Z}_n : \ \text{MCD}(k; n) = 1\} \ ,$$

ed indichiamo con $\phi(n)$ il numero di elementi di $\mathscr{I}_n$. Allora per ogni $x \in \mathscr{I}_n$ si ha

$$x^{\phi(n)} \equiv 1 \mod n. \tag{5.11}$$

La funzione $\phi(n)$ è nota come ϕ di Eulero, ed ha una notevole rilevanza in teoria dei numeri. Se n è primo allora $\phi(n) = n-1$ e in questo caso l'enunciato si riduce a quello del piccolo teorema di Fermat.

5.7. – Dimostrare la formula (5.11) sviluppando la seguente traccia (che ricalca la dimostrazione del piccolo teorema di Fermat): sia X l'insieme dei multipli di x della forma kx con $k \in \mathscr{I}_n$, visti come elementi di $\mathbb{Z}_n$; far vedere che X è contenuto in $\mathscr{I}_n$ ed ha tanti elementi quanti $\mathscr{I}_n$, per cui $X = \mathscr{I}_n$; ne segue che il prodotto dei numeri kx con $k \in \mathscr{I}_n$ è uguale al prodotto dei numeri k in $\mathscr{I}_n$; dedurne la (5.11).

5.8. – Dimostrare che se n è il prodotto di due primi distinti p e q allora $\phi(n) = (p-1)(q-1)$.

5.9. – Dimostrare la seguente generalizzazione della Proposizione 5.3: sia n un prodotto di primi distinti $p_1, \dots, p_h$, ed a un numero intero tale che $\text{MCD}(a; (p_1 - 1) \cdots (p_h - 1)) = 1$. Allora la funzione $f(x) := x^a$ è invertibile su $\mathbb{Z}_n$, e l'inversa è data da $f^{-1}(y) = y^{\bar a}$ dove $\bar a$ è un qualsiasi intero positivo tale che $a\bar a \equiv 1 \mod (p_1 - 1) \cdots (p_h - 1)$.

5.10. – Dimostrare che se $n = p^h$ con p primo ed $h > 1$, allora la funzione $f(x) := x^a$ non è iniettiva per alcun esponente $a > 1$. Suggerimento: calcolare $f(x)$ per $x = p^{h-1}$ ed $x = 1$.

5.11. – Dimostrare che se n non è prodotto di primi distinti allora la funzione $f(x) := x^a$ non è iniettiva per alcun esponente $a > 1$. Suggerimento: sia p un primo tale che p^2 divide n, sia X_1 l'insieme dei numeri $x \in \mathbb{Z}_n$ divisibili per p ed X_2 l'insieme di quelli divisibili per p^2; far vedere che X_1 contiene strettamente X_2, e che f mappa X_1 dentro X_2, e pertanto non può essere iniettiva su X_1.

5.12. Un rafforzamento della Proposizione 5.3 – Se n è il prodotto di due primi distinti p e q, allora l'inversa della funzione $f(x) := x^a$ su $\mathbb{Z}_n$ è data da $f^{-1}(y) = y^{\bar a}$ dove $\bar a$ è un qualsiasi numero intero tale che

$$a\bar a \equiv 1 \mod m \tag{5.12}$$

per un qualche m multiplo comune di $p - 1$ e $q - 1$.

Esaminando la dimostrazione della Proposizione 5.3 si vede infatti che il numero $\bar a$ deve soddisfare $a\bar a \equiv 1 \mod p - 1$ e $a\bar a \equiv 1 \mod q - 1$, il che equivale a dire che $p - 1$ e $q - 1$ dividono $a\bar a - 1$. Chiaramente questo è vero quando m divide $a\bar a - 1$, cioè quando vale la (5.12).

5.13. Un algoritmo per il calcolo delle potenze – Sia a un intero positivo. Osserviamo che le cifre $[a_h, a_{h-1}, \dots, a_1, a_0]$ della rappresentazione di a in base 2 permettono di scrivere a come somma di potenze di due, infatti le cifre a_i sono tutte uguali a 0 o 1 e la formula (1.3) dà

$$a = a_h \cdot 2^h + a_{h-1} \cdot 2^{h-1} + \cdots + a_1 \cdot 2 + a_0 \ .$$

Ne consegue che x^a è il prodotto delle potenze $x^{(a_i 2^i)}$ con $i = 0, 1, \dots, h$, e siccome $x^{(a_i 2^i)} = 1$ quando $a_i = 0$,

$$x^a = \big[\text{prodotto di } x^{(2^i)} \text{ con } i \text{ tale che } a_i = 1\big] \tag{5.13}$$

Questa formula fornisce un algoritmo per il calcolo di x^a alternativo al semplice moltiplicare x per se stesso a volte.

Vogliamo ora stimarne il costo computazionale nel caso che lo si usi per calcolare x^a in $\mathbb{Z}_n$. Per ricavare le cifre della rappresentazione di a in base 2 dobbiamo fare h divisioni per 2 (vedi Sezione 1.11), ciascuna con costo $O(\log a)$ (cfr. Sezione 3.3). Essendo h dell'ordine di $\log a$, il costo complessivo di questa operazione è $O(\log^2 a)$. Calcolare $x^2, x^4, x^8, \ldots, x^{(2^h)}$ significa fare h elevamenti al quadrato consecutivi e siccome ciascun elevamento a quadrato in $\mathbb{Z}_n$ comporta un costo $O(\log^2 n)$ (cfr. Sezione 3.4), il costo complessivo è $O(\log a \log^2 n)$. Infine per ricavare x^a ci servono al più altre h moltiplicazioni, con un ulteriore costo dell'ordine di $O(\log a \log^2 n)$.

In conclusione, supponendo $a \leq n$, il costo dell'algoritmo è $O(\log^3 n)$.

Bibliografia

[1] N. Koblitz, *A Course in Number Theory and Cryptography*, Graduate Texts in Mathematics 114, Springer, New York 1987.

[2] S. Singh, *The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography*, Anchor Books, Londra 1999. Traduzione italiana: *Codici & Segreti*, Rizzoli, Milano 1999.

Giovanni Alberti

Dipartimento di Matematica
Università di Pisa
L.go Pontecorvo 5, I-56127 Pisa, Italy

email: alberti@dm.unipi.it
home page: http://www.dm.unipi.it/~alberti