Dispense del corso di Crittografia
A.A. 2014-2015

Francesco Fumagalli
Università degli Studi di Firenze

Indice

1 Cenni di Teoria dei numeri **5**

1.1 La funzione di Eulero 5

1.2 Campi finiti 7

1.3 Pseudoprimi di Fermat e numeri di Carmichael 8

1.4 La funzione di Carmichael 12

1.5 Residui quadratici 13

1.5.1 Il simbolo di Legendre 13

1.5.2 La legge di reciprocità quadratica 18

1.5.3 Il simbolo di Jacobi 20

1.6 Pseudoprimi di Eulero 23

2 Complessità ed algoritmi **25**

2.1 Cenni di teoria della complessità computazionale 25

2.2 Complessità di alcune operazioni elementari 27

2.3 L'algoritmo di Euclide 31

2.3.1 Algoritmo di Euclide esteso 33

2.3.2 Analisi dei costi computazionali 33

2.4 Estrazioni di radici quadrate in $\mathbb{N}$. 35

2.5 Complessità dell'aritmetica modulare di base. 36

2.6 Estrazioni di radici quadrate in campi finiti. 38

2.7 Equivalenza computazionale 41

3 Sistemi crittografici classici **43**

3.1 Cifrature a blocco 49

3.1.1 Il cifrario di Vigenère 49

3.1.2 Cifrature a blocchi affini lineari 53

3.1.3 Modalità di funzionamento di alcuni cifrari a blocchi 55

4 Crittografia a chiave pubblica **59**
4.1 RSA . . . 62
4.1.1 Osservazioni sulla scelta dei parametri . . . 70
4.2 Il logaritmo discreto . . . 71
4.2.1 Il protocollo Diffie-Hellman-Merkle . . . 71
4.2.2 Il protocollo ElGamal . . . 72
4.2.3 Il protocollo di Massey-Omura . . . 72
4.2.4 Algoritmi per il calcolo del logaritmo discreto . . . 73

5 Firme digitali **79**
5.1 DSA . . . 80
5.2 Van Heyst and Pedersen Fail-stop Signature Scheme . . . 81
5.3 Lo schema di Chaum - Van Antwerpen . . . 84
5.4 Le funzioni Hash . . . 87
5.4.1 Hash di Chaum, van Heyst, Pfitzman . . . 88
5.4.2 Un potenziale attacco alle funzioni Hash . . . 90

6 Protocolli di sicurezza **93**
6.1 Condivisione dei segreti . . . 93
6.1.1 Lo schema a soglia di Shamir . . . 93
6.2 Protocolli a conoscenza zero . . . 95
6.2.1 Il protocollo di Feidge-Fiat-Shamir . . . 96

7 Fattorizzazione di interi **99**
7.1 Test di primalità . . . 99
7.1.1 Eventuali metodi basati sul Teorema di Wilson e sul "piccolo Teorema di Fermat" . . . 99
7.1.2 Il test di Solovay-Strassen . . . 100
7.2 Fattorizzazione di interi . . . 102
7.3 Il metodo ρ di Pollard . . . 103
7.4 Il metodo di Fermat . . . 105
7.5 Il metodo $p-1$ di Pollard . . . 108

Capitolo 1

Cenni di Teoria dei numeri

I numeri interi giocano un ruolo fondamentale nella crittografia moderna. In questo capitolo presentiamo alcuni concetti e risultati chiave su cui si basano i principali protocolli crittografici che vedremo in dettaglio nei capitoli successivi. Si assume che il lettore abbia già una buona familiarità con l'aritmetica modulare e le nozioni di base di teoria dei gruppi, degli anelli e dei campi. In particolare si assumono noti i programmi dei corsi di Algebra 1 e 2, le cui referenze consigliate sono le dispense [9] e [10].

1.1 La funzione di Eulero

Definizione 1. *Dato un intero positivo n, si indica con $\varphi(n)$ il numero di interi compresi tra 1 ed n che sono coprimi con n, ovvero*

$$\varphi(n) := |\{a \in \mathbb{N} | 1 \leq a \leq n \ \ e \ \ (a, n) = 1\}|,$$

dove come d'uso il simbolo (a, n) indica il massimo comune divisore tra gli interi a ed n. La funzione φ così definita si chiama funzione di Eulero.

In particolare $\varphi(1) = 1$ e per $n \geq 2$ è ben noto che $\varphi(n)$ coincide con l'ordine del gruppo moltiplicativo $U(\mathbb{Z}_n)$ degli elementi invertibili dell'anello $\mathbb{Z}_n = \mathbb{Z}/n\mathbb{Z}$ (si veda ad esempio [9], Teorema 6.1). Se $n = p$ è un qualunque numero primo, $\varphi(p) = p - 1$ e, più in generale,

$$\varphi(p^k) = p^k - p^{k-1} = p^k \left(1 - \frac{1}{p}\right), \quad \forall k \geq 1.$$

Questo poiché l'anello $\mathbb{Z}_p$ è un campo, mentre l'anello $\mathbb{Z}_{p^k}$ possiede un unico ideale massimale, precisamente $p\mathbb{Z}/p^k\mathbb{Z}$, il cui ordine è p^{k-1}, dunque gli elementi invertibili di $\mathbb{Z}_{p^k}$ sono tutti i soli elementi che non appartengono a tale ideale. (Oppure più semplicemente basta notare che gli interi primi con p^k sono quelli che non sono multipli di p).

Ricordiamo il seguente importante risultato la cui dimostrazione consiste nel fatto (già citato) che $\varphi(n) = |U(\mathbb{Z}_n)|$.

Teorema 1.1. (Teorema di Eulero-Fermat) *Sia n un intero maggiore di 1. Allora per ogni $a \in \mathbb{Z}$, coprimo con n, vale*

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

In particolare per $n = p$ numero primo, se $a \not| p$ allora $a^{p-1} \equiv 1 \pmod{p}$.

Richiamiamo ora due risultati noti di Algebra 1 (per le dimostrazioni si rimanda alle dispense del primo anno [9]).

Proposizione 1.2. *Siano R ed S due anelli. Allora*

$$U(R \times S) = U(R) \times U(S),$$

dove $U(A)$ indica il gruppo degli elementi invertibili del generico anello A.

Teorema 1.3. (Teorema cinese dei resti) *Siano $n, m \in \mathbb{Z}$ tali che $(n, m) = 1$. Allora*

$$\mathbb{Z}_{mn} \simeq \mathbb{Z}_n \times \mathbb{Z}_m.$$

Proposizione 1.4. *La funzione di Eulero è moltiplicativa, ovvero se $n, m \in \mathbb{N}^*$ e $(n, m) = 1$, allora $\varphi(nm) = \varphi(n)\varphi(m)$.*

Dimostrazione. Per la Proposizione 1.2 ed il Teorema 1.3 abbiamo che

$$\varphi(nm) = |U(\mathbb{Z}_{mn})| = |U(\mathbb{Z}_n \times \mathbb{Z}_m)| = |U(\mathbb{Z}_n)| \cdot |U(\mathbb{Z}_m)| = \varphi(n)\varphi(m).$$

■

Per ogni intero positivo n, otteniamo pertanto una formula per $\varphi(n)$. Basta fattorizzare n come prodotto, in modo essenzialmente unico, di potenze di primi distinti, $n = p_1^{a_1} \dots p_l^{a_l}$, e sfruttate la molteplicità della funzione φ,

$$\varphi(n) = \varphi\left(\prod_{i=1}^{l} p_i^{a_i}\right) = \prod_{i=1}^{l} \varphi(p_i^{a_i}) = \prod_{i=1}^{l} p_i^{a_i}\left(1 - \frac{1}{p_i}\right) = n \cdot \prod_{i=1}^{l}\left(1 - \frac{1}{p_i}\right).$$

Si tratta questa di una formula chiusa ed esplicita. In pratica però per grossi numeri n è di scarsa utilità, in quanto presuppone di conoscere la fattorizzazione di n, cosa non sempre fattibile.

Proposizione 1.5. *Per ogni $n \geq 1$ si ha $\sum_{d|n} \varphi(d) = n$.*

Dimostrazione. Poniamo $[n] = \{1, 2, \ldots, n\}$ e $D_n = \{d | 1 \leq d \leq n,\ d|n\}$ e definiamo l'applicazione

$$\begin{aligned} \theta : [n] &\longrightarrow D_n \\ a &\longmapsto (a, n). \end{aligned}$$

Allora

$$n = \sum_{d|n} \left|\theta^{-1}(d)\right|.$$

D'altra parte

$$\left|\theta^{-1}(d)\right| = |\{a | 1 \leq a \leq n,\ (a, n) = d\}| = |\{b | 1 \leq b \leq n/d,\ (b, n/d) = 1\}| = \varphi(n/d).$$

Dunque

$$\sum_{d|n} \varphi(d) = \sum_{d|n} \varphi(n/d) = \sum_{d|n} \left|\theta^{-1}(d)\right| = n.$$

■

1.2 Campi finiti

Incominciamo col ricordare il Teorema di esistenza ed unicità (a meno di isomorfismi) per i campi finiti. Si rimanda a [10], Teorema 7.5.

Teorema 1.6. *Sia p un primo e sia $n \in \mathbb{N}$, $n \geq 1$. Allora esiste un campo finito di ordine p^n. Inoltre ogni campo finito ha ordine una potenza di qualche numero primo p e due campi finiti dello stesso ordine sono isomorfi.*

Proviamo ora che il gruppo moltiplicativo di ogni campo finito è un gruppo ciclico. Si confronti questo risultato con il Teorema 7.6 in [10] il cui enunciato è leggermente più generale.

Teorema 1.7. *Il gruppo moltiplicativo $\mathbb{F}^*$ di un campo finito $\mathbb{F}$ è ciclico.*

Per la dimostrazione facciamo uso del seguente Lemma, dovuto a J.P. Serre.

Lemma 1.8. *Sia G un gruppo finito d'ordine n. Se per ogni $k \geq 1$ il sottoinsieme $\{g \in G | g^k = 1\}$ ha cardinalità al più k, allora G è un gruppo ciclico.*

Dimostrazione. Per ogni intero k definiamo i seguenti sottoinsiemi di G.

$$A_k(G) = \{g \in G | g^k = 1\} \quad \text{e} \quad B_k(G) = \{g \in G |\, |g| = k\}.$$

Per ipotesi abbiamo che per ogni $k \geq 1$, $|A_k(G)| \leq k$. Inoltre ovviamente $B_k(G) \subseteq A_k(G)$. Supponiamo che per un certo k si abbia $B_k(G) \neq \varnothing$. Allora per il teorema di Lagrange, $k|n$. Preso un qualunque elemento $x \in B_k(G)$, il sottogruppo $\langle x \rangle$ ha ordine k, pertanto $\langle x \rangle \simeq \mathbb{Z}_k$ e contiene esattamente $\varphi(k)$ elementi di ordine k. Ne segue che $|B_k(G)| \geq \varphi(k)$. Affermiamo che $B_k(G) \subseteq \langle x \rangle$. Se così non fosse, preso un $y \in B_k(G) \setminus \langle x \rangle$ si avrebbe che

$$|A_k(G)| \geq |\langle x \rangle \cup \langle y \rangle| \geq k + 1,$$

contro l'ipotesi dell'enunciato. Pertanto $B_k(G) \subseteq \langle x \rangle$, in particolare abbiamo che

$$\varphi(k) \leq |B_k(G)| \leq |B_k(\langle x \rangle)| = \varphi(k),$$

da cui segue che se $B_k(G) \neq \varnothing$, allora $k|n$ e $|B_k(G)| = \varphi(k)$. Ora poiché ogni elemento di G ha un ben preciso ordine che è un divisore di n,

$$G = \bigsqcup_{k|n} B_k(G)$$

(dove il simbolo $\bigsqcup$ indica che l'unione è disgiunta). Passando alle cardinalità ed utilizzando la Proposizione 1.5, abbiamo che

$$n = \sum_{k|n} |B_k(G)| \leq \sum_{k|n} \varphi(k) = n,$$

che è soddisfatta se e solo se per ogni $k|n$ si ha $B_k(G) \neq \varnothing$. In particolare per $k = n$, G possiede elementi d'ordine n, cioè G è ciclico. ■

Dimostrazione del Teorema 1.7.
$\mathbb{F}^*$ è un gruppo abeliano finito, inoltre per ogni $k \geq 1$ il polinomio $x^k - 1$ ha al più k radici in un suo campo di spezzamento su $\mathbb{F}$, ovvero, per ogni $k \geq 1$, $|\{\alpha \in \mathbb{F}^* | \alpha^k = 1\}| \leq k$. Per il Lemma precedente segue che $\mathbb{F}^*$ è un gruppo ciclico. ■

1.3 Pseudoprimi di Fermat e numeri di Carmichael

Definizione 2. *Sia n un intero dispari e sia a un intero coprimo con n. Chiameremo n* pseudoprimo di Fermat rispetto alla base *a se n è un numero composto e $a^{n-1} \equiv 1$ (mod n).*

Per il Teorema 1.1 è evidente che se n è un numero primo allora $a^{n-1} \equiv 1 \pmod{n}$, per ogni a coprimo con n. Non vale però il viceversa, ovvero la definizione 2 è ben posta. Lasciamo infatti verificare al lettore che il numero $341 = 11 \cdot 31$ è pseudoprimo di Fermat rispetto a 2. Forse ancora più è il fatto che esistono numeri composti che sono pseudoprimi rispetto a qualunque base. Tali numeri vengono detti di Carmichael.

Definizione 3. *Un numero intero n che è pseudoprimo rispetto ad ogni base a, $(a, n) = 1$, si dice* numero di Carmichael.

I più piccoli numeri di questo tipo sono:

$$561 = 3 \cdot 11 \cdot 17, \quad 1105 = 5 \cdot 13 \cdot 17, \quad 1729 = 7 \cdot 13 \cdot 19^1$$

Nel 1939 J. Chernick ha dimostrato che al variare di k il numero

$$(6k + 1)(12k + 1)(18k + 1)$$

è di Carmichael se tutti e tre i fattori sono numeri primi ([12]). Se questa formula sia in grado di fornire un'infinità di numeri di Carmichael è tuttora una questione aperta. Recentemente però W. R. Alford, Andrew Granville e Carl Pomerance hanno dimostrato che esistono infiniti numeri di Carmichael. Più precisamente hanno provato che per n sufficientemente grande ci sono almeno $n^{2/7}$ numeri di Carmichael compresi tra 1 ed n (si veda [2]). Esiste altresì un'importante caratterizzazione di questi numeri, dovuta a Korselt (1899), che ora andiamo a provare.

Teorema 1.9. *Un numero naturale composto n è di Carmichael se e solo se n è libero da quadrati e per ogni divisore primo p di n si ha che $p - 1$ divide $n - 1$.*

Per la dimostrazione ci occorre il seguente

Lemma 1.10. *Sia p un numero primo dispari. Allora per ogni $k \geq 1$ il gruppo moltiplicativo $U\left(\mathbb{Z}_{p^k}\right)$ è ciclico.*

Dimostrazione. Poniamo $U = U\left(\mathbb{Z}_{p^k}\right)$. Abbiamo che $|U| = \varphi(p^k) = p^{k-1}(p - 1)$. Mostreremo che U è prodotto diretto di due suoi sottogruppi ciclici, P e Q, di ordine rispettivamente p^{k-1} e $p-1$, essendo $(p^{k-1}, p-1) = 1$, la tesi seguirà in modo immediato. Consideriamo l'epimorfismo di anelli

$$\begin{aligned} \pi : \mathbb{Z}_{p^k} &\longrightarrow \mathbb{Z}_p \\ a + p^k\mathbb{Z} &\longmapsto a + p\mathbb{Z}. \end{aligned}$$

π induce un epimorfismo tra i rispettivi gruppi moltiplicativi, che chiamiamo

$$\phi = \pi_{|U} : U(\mathbb{Z}_{p^k}) = U \longrightarrow U(\mathbb{Z}_p) = \mathbb{Z}_p^*.$$

[1] ≪I remember once going to see him when he was ill at Putney. I had ridden in taxi cab number 1729 and remarked that the number seemed to me rather a dull one, and that I hoped it was not an unfavorable omen. "No," he replied, "it is a very interesting number; it is the smallest number expressible as the sum of two cubes in two different ways [$1729 = 1^3 + 12^3 = 9^3 + 10^3$]"≫. ([16])

Sia $P = \ker(\phi)$. Per il primo Teorema di omomorfismo fra gruppi si ha che $U/P \simeq \mathbb{Z}_p^*$, da cui segue

$$|P| = \frac{|U|}{|\mathbb{Z}_p^*|} = \frac{p^{k-1}(p-1)}{p-1} = p^{k-1}.$$

Essendo $p-1$ coprimo con p, P è un p-sottogruppo di Sylow di U e poiché U è abeliano, P è l'unico p-Sylow di U. Sempre dal fatto che U è abeliano segue che U possiede un unico sottogruppo Q d'ordine $p-1$ (Q è esattamente prodotto diretto interno dei sottogruppi di Sylow di U associati ai primi diversi da p). È evidente che $U = P \times Q$. Inoltre applicando i Teoremi di omomorfismo fra gruppi otteniamo

$$Q \simeq \frac{Q}{P \cap Q} \simeq \frac{PQ}{P} = \frac{U}{P} \simeq \phi(U) = \mathbb{Z}_p^*.$$

In particolare Q è ciclico per il Teorema 1.7. Da ultimo ci resta da prova re che P è ciclico. A tal fine si consideri la classe $1 + p + p^k\mathbb{Z} \in U$. Sviluppando il binomio di Newton abbiamo per ogni $h \geq 1$

$$\begin{aligned}(1+p)^{p^{h-1}} = \sum_{j=0}^{p^{h-1}} \binom{p^{h-1}}{j} p^j &\equiv 1 \pmod{p^h} \quad \text{e} \\ &\equiv 1 + p^h \not\equiv 1 \pmod{p^{h+1}}.\end{aligned}$$

Da cui segue che $\left|1 + p + p^k\mathbb{Z}\right| = p^{k-1}$, e quindi il sottogruppo P risulta essere ciclico generato da $1 + p + p^k\mathbb{Z}$. ■

La dimostrazione del Teorema 1.9 fa anche uso del (già citato) Teorema cinese dei resti che questa volta enunciamo nella seguente versione (vd [9], Teorema 4.13).

Teorema 1.11. (Teorema cinese dei resti). *Siano $n_1, n_2, \ldots, n_k$ interi a due a due coprimi. Allora comunque si scelgano k interi $a_1, a_2, \ldots, a_k$ esiste un intero x soluzione del seguente sistema di congruenze*

$$\{x_i \equiv a_1 \pmod{n_1}$$

Inoltre tutte le soluzioni del sistema sono congrue modulo $n = n_1 n_2 \ldots n_k$.

Dimostrazione del Teorema 1.9.
Sia n un numero di Carmichael. Mostriamo innanzitutto che n è privo di quadrati. Per assurdo sia $n = p^k m$, con p primo, $(p, m) = 1$ e $k \geq 2$. Sia $b + p^k\mathbb{Z}$ un generatore del gruppo ciclico $U\left(\mathbb{Z}_{p^k}\right)$, che sappiamo esistere per il Lemma 1.10. Per il Teorema cinese dei resti, esiste un $a \in \mathbb{Z}$ tale che

$$\begin{cases} a \equiv b \pmod{p^k} \\ a \equiv 1 \pmod{m} \end{cases}$$

In particolare abbiamo che a è coprimo sia con p^k che con m, e dunque $(n, a) = 1$. Essendo per ipotesi n di Carmicael, $n|a^{n-1} - 1$, e quindi anche $p^k|a^{n-1} - 1$. Ora $b^{n-1} \equiv a^{n-1} \equiv 1 \pmod{p^k}$, pertanto

$$\left|b + p^k\mathbb{Z}\right| = \varphi(p^k) = p^{k-1}(p-1)|n-1.$$

In particolare $p|n-1$, il che è assurdo essendo p un divisore primo dispari di n.
Proviamo ora che se p è un primo che divide n allora $p-1|n-1$. A tal fine sia $n = pm$, con $(m, p) = 1$ per quanto appena osservato sopra. Sia $c + p\mathbb{Z}$ un generatore di $\mathbb{Z}_p^*$ e sia $a \in \mathbb{Z}$ una soluzione del sistema di congruenze

$$\begin{cases} x \equiv c \pmod{p} \\ x \equiv 1 \pmod{m} \end{cases}$$

che sappiamo esistere per il Teorema 1.11. Abbiamo allora che a ed n sono coprimi, dunque $a^{n-1} \equiv 1 \pmod{n}$. Pertanto $c^{n-1} \equiv a^{n-1} \equiv 1 \pmod{p}$, cioè $|c + p\mathbb{Z}| = p-1|n-1$, il che è assurdo essendo p un divisore di n.

Proviamo ora l'altra implicazione. Sia n un intero naturale libero da quadrati tale che per ogni primo p di n, $p-1$ divida $n-1$. Scritto $n = \prod_{i=1}^{l} p_i$, con p_i primi distinti, l'applicazione

$$\begin{aligned} f : \mathbb{Z}_n &\longrightarrow \mathbb{Z}_{p_1} \times \ldots \times \mathbb{Z}_{p_l} \\ z + n\mathbb{Z} &\longmapsto (z + p_1\mathbb{Z}, \ldots, z + p_l\mathbb{Z}) \end{aligned}$$

è un isomorfismo di anelli (Teorema 1.3). In particolare preso arbitrariamente $a \in \mathbb{Z}$ coprimo con n, si ha che

$$f(a^{n-1} + n\mathbb{Z}) = (a^{n-1} + p_1\mathbb{Z}, \ldots, a^{n-1} + p_l\mathbb{Z}) = (1 + p_1\mathbb{Z}, \ldots, 1 + p_l\mathbb{Z}) = 1_{\mathbb{Z}_{p_1} \times \ldots \times \mathbb{Z}_{p_l}},$$

essendo $p_i - 1$ un divisore di $n-1$ per ogni i. Poiché f è un isomorfismo, $a^{n-1} + n\mathbb{Z} = 1 + n\mathbb{Z} = 1_{\mathbb{Z}_n}$, ovvero $a^{n-1} \equiv 1 \pmod{n}$ ed n è pseudoprimo di Fermat rispetto ad a. ■

Corollario 1.12. *Sia n un numero di Carmichael. Allora n ha almeno tre divisori primi distinti.*

Dimostrazione. Supponiamo per assurdo che sia $n = pq$, con p e q primi distinti $p < q$. Per il Teorema 1.9 abbiamo che $q-1|n-1$. Ne segue che

$$p - 1 \equiv p(q-1) + p - 1 = pq - 1 = n - 1 \equiv 0 \pmod{q-1}.$$

Ovvero $q-1|p-1$, il che è assurdo essendo $p < q$. ■

1.4 La funzione di Carmichael

Definizione 4. *Dato $n \in \mathbb{N}$ indichiamo con $\lambda(n)$ il più piccolo intero positivo m per cui $a^m \equiv 1$ (mod n) per ogni intero a coprimo con n. La funzione $\lambda(n)$ dice funzione di Carmichael.*

L'intero $\lambda(n)$ rappresenta quello che in Teoria dei Gruppi si chiama *esponente* del gruppo $U(\mathbb{Z}_n)$, ovvero il più piccolo intero naturale m per cui $\alpha^m = 1$, per ogni $\alpha \in U(\mathbb{Z}_n)$. Essendo $U(\mathbb{Z}_n)$ un gruppo d'ordine $\varphi(n)$ è evidente che

$$\lambda(n)|\varphi(n). \tag{1.1}$$

I due numeri sono però in generale diversi, ad esempio già per $n = 8$ abbiamo che $\lambda(8) = 2$ e $\varphi(8) = 4$.

Esercizio 1.1. Si provi la relazione (1.1).

Esercizio 1.2. Si provi che $\varphi(35) = 24$ e $\lambda(35) = 12$.

La seguente Proposizione consente di calcolare agevolmente la funzione di Carmichael di un numero intero, una volta nota la sua fattorizzazione.

Proposizione 1.13. *1. Se p è un primo dispari e $k > 0$ allora*

$$\lambda(p^k) = \varphi(p^k) = p^k \left(1 - \frac{1}{p}\right).$$

2. $\lambda(2) = 1$, $\lambda(4) = 2$ e $\lambda(2^k) = 2^{k-2}$ per ogni $k \geq 3$.

3. Se $n = \prod_{i=1}^{l} p_i^{a_i}$ con p_i primi distinti ed $a_i > 0$ per ogni $i = 1, \ldots, l$, allora

$$\lambda(n) = m.c.m\left\{\lambda(p_i^{a_i}) \,|\, i = 1, 2, \ldots, l\right\}.$$

Dimostrazione. 1. segue immediatamente dal Lemma 1.10.
2. è banalmente vera per i casi 2^k con $k = 0, 1$. Sia $k \geq 3$, proviamo nell'ordine le seguenti affermazioni

2.1. $\lambda(2^k)|\frac{\varphi(2^k)}{2}$.

2.2. Se $x \equiv 3 \pmod 8$ o $x \equiv 5 \pmod 8$, allora $\left|x + 2^k\mathbb{Z}\right| = \frac{\varphi(2^k)}{2} = 2^{k-2}$.

2.1. si prova per induzione su k. Per $k = 3$ è immediato verificare che $y^2 \equiv 1 \pmod 8$ per ogni intero dispari y. Assunto per ipotesi induttiva che $y^{2^{k-2}} \equiv 1 \pmod{2^k}$ e scritto $y = 1 + t2^k$, si ha che

$$y^{2^{k-1}} = (y^{2^{k-2}})^2 = (1 + t2^k)^2 = 1 + t2^{k+1} + t^2 2^{2k} \equiv 1 \pmod{2^{k+1}}.$$

2.2. Facciamo induzione su k. Per $k = 3$ è banale. Assumiamo il risultato vero per ogni $h \leq k$ e proviamolo per $k+1$. In particolare, preso x come nell'enunciato b., abbiamo che

$$\left|x + 2^{k-1}\mathbb{Z}\right| = 2^{k-3} \quad \text{e} \quad \left|x + 2^{k}\mathbb{Z}\right| = 2^{k-2},$$

che implicano

$$\begin{aligned} x^{2^{k-3}} &\equiv 1 \,(\text{mod } 2^{k-1}) \\ &\not\equiv 1 \,(\text{mod } 2^{k}) \end{aligned}$$

da cui segue necessariamente che $x^{2^{k-3}} \equiv 1 + 2^{k-1} \pmod{2^k}$. Pertanto

$$x^{2^{k-2}} = (x^{2^{k-3}})^2 \equiv (1 + 2^{k-1})^2 \equiv 1 + 2^k \not\equiv 1 \pmod{2^{k+1}}.$$

Abbiamo pertanto provato che $\left|x + 2^{k+1}\mathbb{Z}\right| > 2^{k-2}$. D'altra parte per il punto 2.1. sappiamo che $\left|x + 2^{k+1}\mathbb{Z}\right|$ divide $\frac{\varphi(2^{k+1})}{2} = 2^{k-1}$, ne risulta quindi che $\left|x + 2^{k+1}\mathbb{Z}\right| = 2^{k-1}$, e il punto 2. è così dimostrato.
3. Per i punti 1. e 2. appena provati è sufficiente mostrare che se m ed n sono due interi coprimi allora $\lambda(mn) = m.c.m.\{\lambda(m), \lambda(n)\}$. Detto $x = m.c.m.\{\lambda(m), \lambda(n)\}$ e preso un arbitrario $a \in \mathbb{Z}$ coprimo con mn, si ha che $a^{\lambda(mn)} \equiv 1 \pmod{mn}$, pertanto anche

$$\begin{cases} a^{\lambda(mn)} \equiv 1 \pmod{m} \\ a^{\lambda(mn)} \equiv 1 \pmod{n} \end{cases}$$

ovvero $\lambda(m)|\lambda(mn)$ e $\lambda(n)|\lambda(mn)$, da cui segue che $x|\lambda(mn)$. Viceversa, dal fatto che m ed n sono coprimi e $\lambda(m)|x$, $\lambda(n)|x$ segue che per ogni a coprimo con mn, $a^x \equiv 1 \pmod{mn}$. Quindi $\lambda(mn) \leq x$ per come è definito. Se ne deduce che $\lambda(mn) = x$. ■

1.5 Residui quadratici

1.5.1 Il simbolo di Legendre

In molte situazioni è utile saper risolvere congruenze del tipo

$$x^2 \equiv d \,(\text{mod } p),$$

dove d è un intero fissato e p un primo che non divide d. Ad esempio ogni congruenza di secondo grado

$$ax^2 + bx + c \equiv 0 \,(\text{mod } p)$$

(con $a, b, c \in \mathbb{Z}$) ammette soluzioni se e solo se $\Delta = \bar{b}^2 - 4\bar{a}\bar{c}$ è un quadrato in $\mathbb{Z}_p$, ovvero se e solo se

$$x^2 \equiv \Delta \,(\text{mod } p)$$

è risolvibile.

Definizione 5. *Siano $n, a \in \mathbb{Z}$, con $n \neq 1$. L'intero a si dice* residuo quadratico modulo *n (in breve,* R.Q.*) se esiste $b \in \mathbb{Z}$ tale che*

$$b^2 \equiv a \,(mod\ n).$$

Il caso particolarmente interessante è quando n è un numero primo dispari p. Se $n = 2$ allora ogni elemento di $\mathbb{Z}/2\mathbb{Z}$ è un quadrato, quindi ogni intero è un residuo quadratico modulo 2. Se invece p è un primo dispari allora la questione si fa interessante e conviene introdurre la seguente

Definizione 6. *Sia p un primo dispari ed a un intero. Definiamo il* simbolo di Legendre *come segue*

$$\left(\frac{a}{p}\right) = \begin{cases} 0 \ se\ p|a, \\ 1 \ se\ p \not| a \ ed\ a\ è\ un\ R.Q.\ modulo\ p, \\ -1 \ se\ a\ non\ è\ un\ R.\ Q.\ modulo\ p. \end{cases}$$

Sia p un primo dispari e indichiamo con G il gruppo moltiplicativo $(\mathbb{Z}_p)^*$. Sia α un generatore di G. Poiché 2 divide $p-1$, $Q = \langle \alpha^2 \rangle$ è un sottogruppo di G di ordine $\frac{p-1}{2}$. È facile vedere che Q è l'insieme dei quadrati non nulli di $\mathbb{Z}_p$. Sia infatti x un elemento di Q, allora esiste un n, $0 \leq n \leq p-2$, tale che

$$x = (\alpha^2)^n = (\alpha^n)^2$$

e quindi x è un quadrato. Viceversa, preso un quadrato $y \neq 0$ in $\mathbb{Z}_p$, questo è un quadrato di un elemento di G, dunque, per qualche intero k,

$$y = (\alpha^k)^2 = (\alpha^2)^k \in Q.$$

In particolare si ha che esattamente la metà degli elementi non nulli di $\mathbb{Z}_p$ è un quadrato; contando anche lo 0, il numero dei quadrati di $\mathbb{Z}_p$ è $1 + (p-1)/2 = (p+1)/2$.

Proposizione 1.14. *(Criterio di Eulero) Siano p un primo dispari ed $a \in \mathbb{Z}$ con $(a, p) = 1$. Allora*

$$\left(\frac{a}{p}\right) \equiv a^{\frac{p-1}{2}} \,(mod\ p).$$

Dimostrazione. Siano p un primo dispari ed $a \in \mathbb{Z}$ con $(a,p)=1$. Per il Teorema di Fermat, $a^{\frac{p-1}{2}}$ è una soluzione della congruenza

$$x^2 \equiv 1 \pmod{p},$$

e poiché tale equazione ha esattamente due soluzioni modulo p, che sono ± 1, si ha che

$$a^{\frac{p-1}{2}} \equiv \pm 1 \pmod{p}.$$

Ora $\left(\frac{a}{p}\right) = 1$ se a è un R.Q. modulo p, ovvero se esiste $b \in \mathbb{Z}$ tale che $b^2 \equiv a \pmod{p}$. Pertanto, ancora per Fermat,

$$a^{\frac{p-1}{2}} \equiv b^{p-1} \equiv 1 \pmod{p}.$$

Per quanto detto prima, il numero dei quadrati non nulli modulo p è $\frac{p-1}{2}$, e questi sono tutte e sole le radici $(p-1)/2$-esime dell'unità, ovvero tutte e sole le soluzioni dell'equazione di $\mathbb{Z}_p$:

$$x^{\frac{p-1}{2}} = \bar{1}.$$

Ne segue che se a non è un R.Q. modulo p, allora $\bar{a}$ non è una soluzione dell'equazione di sopra, e pertanto, per quanto detto precedentemente, deve essere

$$a^{\frac{p-1}{2}} \equiv -1 \pmod{p},$$

il che prova la Proposizione. ■

Lemma 1.15. *Siano p un primo dispari ed $a, b \in \mathbb{Z}$. Allora*

1. *se $a \equiv b$ (mod p), allora $\left(\frac{a}{p}\right) = \left(\frac{b}{p}\right)$;*
2. $\left(\frac{ab}{p}\right) = \left(\frac{a}{p}\right)\left(\frac{b}{p}\right)$;
3. $\left(\frac{a^2}{p}\right) = 1$;
4. $\left(\frac{-1}{p}\right) = (-1)^{\frac{p-1}{2}}$.

Dimostrazione. I punti 1. e 3. discendono immediatamente dalla definizione. 2. è ovvia se p divide ab. Altrimenti, per il criterio di Eulero, abbiamo

$$\left(\frac{ab}{p}\right) \equiv (ab)^{\frac{p-1}{2}} = a^{\frac{p-1}{2}} b^{\frac{p-1}{2}} \equiv \left(\frac{a}{p}\right)\left(\frac{b}{p}\right) \pmod{p}.$$

Infine, per il criterio di Eulero (Proposizione 1.14) vale

$$\left(\frac{-1}{p}\right) \equiv (-1)^{\frac{p-1}{2}} \pmod{p},$$

ovvero 4. ■

Vediamo di sviluppare una procedura per il calcolo di $\left(\frac{a}{p}\right)$.

Proposizione 1.16. *Sia p un primo dispari. Allora*

$$\left(\frac{2}{p}\right) = (-1)^{\frac{p^2-1}{8}}.$$

Dimostrazione. Osserviamo preliminarmente che esiste un'unica funzione f da $\mathbb{N}^*$ in $\{-1, 1\}$ con le seguenti proprietà:

- $f(2) = 0$,
- $f(p) = (-1)^{\frac{p^2-1}{8}}$ per ogni p primo dispari,
- f è completamente moltiplicativa, cioè $f(mn) = f(m)f(n)$, $\forall m, n \in \mathbb{N}^*$.

Tale funzione è definita da

$$f(n) = \begin{cases} (-1)^{\frac{n^2-1}{8}} & \text{se } n \text{ è dispari} \\ 0 & \text{se } n \text{ è pari} \end{cases}$$

Che questa sia moltiplicativa segue dal fatto che per ogni n intero dispari, $n^2 - 1 \equiv 0 \pmod{8}$ e quindi

$$\begin{aligned} \frac{m^2n^2-1}{8} &= \frac{m^2-1}{8} + \frac{n^2-1}{8} + \frac{(m^2-1)(n^2-1)}{8} \\ &\equiv \frac{m^2-1}{8} + \frac{n^2-1}{8} \pmod{8}. \end{aligned}$$

Proviamo che $f(p) = \left(\frac{2}{p}\right)$ per ogni primo $p \neq 2$.
Consideriamo un'estensione del campo $\mathbb{Z}_p$ che contenga una radice primitiva ottava, ϵ. Allora ϵ è radice di $x^8 - 1 = (x^4 - 1)(x^4 + 1)$ ed essendo primitiva non è radice di $x^4 - 1$. In particolare $\epsilon^4 = -1$. Definiamo la somma di Gauss come segue

$$G = \sum_{j=\bar{0}}^{\bar{7}} \overline{f(j)}\epsilon^j = \epsilon - \epsilon^3 - \epsilon^5 + \epsilon^7,$$

dove con $\overline{f(j)}$ abbiamo indicato la classe $f(j)+p\mathbb{Z} \in \mathbb{Z}_p$. Notiamo preliminarmente che se $j \equiv k \pmod 8$ allora $f(j) = f(k)$ (lo si dimostri per esercizio). Ne segue che se $j \equiv k \pmod 8$ allora

$$\overline{f(j)}\epsilon^j = \overline{f(k)}\epsilon^k,$$

e pertanto per ogni I sistema di rappresentanti per le classi di congruenza modulo 8 si ha che

$$G = \sum_{i \in I} \overline{f(i)}\epsilon^i. \tag{1.2}$$

Ora abbiamo che

$$G = \bar{2}(\epsilon - \epsilon^3) \quad \text{e} \quad G^2 = \bar{4}(\epsilon^2 - \bar{2}\epsilon^4 + \epsilon^6) = \bar{8}.$$

Calcoliamo G^p in due modi diversi. Innanzitutto

$$G^p = (G^2)^{\frac{p-1}{2}} G = \bar{8}^{\frac{p-1}{2}} G = \overline{\left(\frac{8}{p}\right)} G = \overline{\left(\frac{2}{p}\right)} G, \tag{1.3}$$

dove negli ultimi due passaggi si sono usate la Proposizione 1.14 ed il Lemma 1.15. Inoltre per l'automorfismo di Frobenius si ha che

$$G^p = \sum_{j=\bar{0}}^{\bar{7}} \overline{f(j)^p}\epsilon^{pj}.$$

Ora $\overline{f(j)^p} = \overline{f(j)}$ per il Teorema di Fermat, ed essendo $p^2 \equiv 1 \pmod 8$, per quanto notato sopra $f(j) = f(p^2 j)$, perció passando al campo $\mathbb{Z}_p$:

$$\overline{f(j)} = \overline{f(p^2 j)} = \overline{f(p)} \cdot \overline{f(pj)}.$$

Ne segue che

$$G^p = \overline{f(p)} \sum_{j=\bar{0}}^{\bar{7}} \overline{f(pj)}\epsilon^{pj} = \overline{f(p)}G, \tag{1.4}$$

dove nell'ultimo passaggio abbiamo usato la (1.2). Dal confronto di (1.3) con (1.4), si deduce che

$$\overline{\left(\frac{2}{p}\right)} = \overline{f(p)},$$

ovvero p divide $\left(\frac{2}{p}\right) - f(p)$. Per come sono definiti i simboli di Legendre e la funzione f, è evidente che $\left|\left(\frac{2}{p}\right) - f(p)\right| \leq 2$, dunque, essendo p un primo dispari, $\left(\frac{2}{p}\right) = f(p) = (-1)^{\frac{p^2-1}{8}}$, come volevasi dimostrare. ■

1.5.2 La legge di reciprocità quadratica

La Legge di reciprocità quadratica, che enunciamo e proviamo in questa sezione, fu congetturata da Eulero e Legendre, ma provata in modo completo solo da Gauss. Egli la definì *Aureum Theorema* e negli anni ne pubblicò svariate dimostrazioni. Il libro di Franz Lemmermeyer *Reciprocity Laws: From Euler to Eisenstein*, [21], contiene ben 196 dimostrazioni differenti. Al momento (Marzo 2015) ne sono note più di 210.

Lemma 1.17. *Siano p un primo dispari ed A un sistema di rappresentanti delle classi di congruenza modulo p. Allora $\sum_{a\in A}\left(\frac{a}{p}\right)=0$.*

Dimostrazione. Essendo p dispari, esattamente metà degli elementi di $\mathbb{Z}_p^*$ è un quadrato e ciascuno di essi fornisce un contributo di $+1$ alla somma; la restante metà degli elementi di $\mathbb{Z}_p^*$ sono non quadrati e contribuiscono con -1 alla somma, mentre la classe nulla dà contributo 0. ■

Teorema 1.18. *(Legge di reciprocità quadratica di Gauss) Siano p, q due primi dispari distinti. Allora*

$$\left(\frac{p}{q}\right)=(-1)^{\frac{p-1}{2}\frac{q-1}{2}}\left(\frac{q}{p}\right)=\begin{cases}-\left(\frac{q}{p}\right) & \text{se } p\equiv q\equiv 3\,(mod\,4)\\ \left(\frac{q}{p}\right) & \text{altrimenti.}\end{cases}.$$

Dimostrazione. Siano p e q due primi dispari distinti. Nel seguito per facilitare la scrittura indichiamo con $\mathbb{F}$ il campo $\mathbb{Z}_p$. Sia $\mathbb{E}$ il campo di spezzamento del polinomio x^p-1 su $\mathbb{Z}_q$ e sia $\omega\in\mathbb{E}$ una radice primitiva p-esima. Pertanto, $\left\{1,\omega,\omega^2,\ldots,\omega^{p-1}\right\}$ sono tutte e sole le radici distinte del polinomio $x^p-1=(x-1)(x^{p-1}+\ldots+x+1)$ nel campo $\mathbb{E}$. In particolare

$$\omega^{p-1}+\omega^{p-2}\ldots+\omega+1=0. \tag{1.5}$$

Osserviamo che per ogni $z\in\mathbb{Z}$, il valore di ω^z dipende solo dalla classe di congruenza di z modulo p, infatti per ogni $z,m\in\mathbb{Z}$ si ha $\omega^{z+mp}=\omega^z(\omega^p)^m=\omega^z\cdot 1=\omega^z$. Ha senso pertanto definire la potenza ω^a per ogni $a\in\mathbb{F}$. Notiamo che per ogni $a\in\mathbb{F}^*$, l'applicazione $y\mapsto ay$, al variare di $y\in\mathbb{F}^*$, è una permutazione di $\mathbb{F}^*$, con inversa $y\mapsto a^{-1}y$. Da (1.5) segue che per ogni $a\in\mathbb{F}^*$, vale

$$\sum_{x\in\mathbb{F}^*}\omega^{ax}=\sum_{x\in\mathbb{F}^*}\omega^x=\omega+\omega^2+\ldots+\omega^{p-1}=-1. \tag{1.6}$$

Nel seguito adotteremo un abuso di scrittura ed indicheremo gli elementi di $\mathbb{Z}_q$ semplicemente con gli interi naturali tra 0 e $q-1$, omettendo il soprassegno, ovvero, se a è un intero, scriveremo semplicemente a per indicare l'elemento di $\mathbb{Z}_q$ che corrisponde ad $\overline{a}=a+q\mathbb{Z}$. Alla fine della dimostrazione troveremo una relazione che vale in $\mathbb{Z}_q$

e da questa risaliremo alla condizione fra interi dell'enunciato. Definiamo la seguente applicazione

$$\begin{aligned} \tau : \mathbb{F} &\longrightarrow \mathbb{E} \\ a &\longmapsto \sum_{x\in\mathbb{F}^*} \left(\frac{x}{p}\right) \omega^{ax}. \end{aligned}$$

Proviamo che per ogni $a \in \mathbb{F}^*$,

$$\tau(a) = \left(\frac{a}{p}\right) \tau(1). \tag{1.7}$$

Infatti,

$$\begin{aligned} \tau(a) &= \sum_{x\in\mathbb{F}^*} \left(\frac{x}{p}\right) \omega^{ax} = \sum_{x\in\mathbb{F}^*} \left(\frac{a^{-1}xa}{p}\right) \omega^{ax} = \left(\frac{a^{-1}}{p}\right) \sum_{x\in\mathbb{F}^*} \left(\frac{xa}{p}\right) \omega^{ax} = \\ &= \left(\frac{a}{p}\right) \sum_{y\in\mathbb{F}^*} \left(\frac{y}{p}\right) \omega^{y} = \left(\frac{a}{p}\right) \tau(1). \end{aligned}$$

Ora proviamo che

$$(\tau(1))^2 = (-1)^{\frac{p-1}{2}} p. \tag{1.8}$$

Infatti, applicando (1.7), si ha

$$\begin{aligned} \tau(1)^2 &= \tau(1) \sum_{x\in\mathbb{F}^*} \left(\frac{x}{p}\right) \omega^{x} = \sum_{x\in\mathbb{F}^*} \tau(x)\omega^{x} = \sum_{x\in\mathbb{F}^*} \left[\sum_{y\in\mathbb{F}^*} \left(\frac{y}{p}\right) \omega^{xy}\right] \omega^{x} = \\ &= \sum_{x\in\mathbb{F}^*} \sum_{y\in\mathbb{F}^*} \left(\frac{y}{p}\right) \omega^{x(y+1)} = \sum_{y\in\mathbb{F}^*} \left[\left(\frac{y}{p}\right) \sum_{x\in\mathbb{F}^*} \omega^{x(y+1)}\right]. \end{aligned}$$

Ora, dalla formula (1.6), se $y \in \mathbb{F}^*$ e $y + 1 \neq 0$, si ha

$$\sum_{x\in\mathbb{F}^*} \omega^{x(y+1)} = -1.$$

Quindi, applicando i Lemmi 1.17 e 1.15,

$$\begin{aligned} \tau(1)^2 &= -\sum_{y\in\mathbb{F}^*,\, y\neq 1} \left(\frac{y}{p}\right) + \left(\frac{-1}{p}\right) \sum_{y\in\mathbb{F}^*} \omega^0 = \\ &- \sum_{y\in\mathbb{F}^*} \left(\frac{y}{p}\right) + \left(\frac{-1}{p}\right) + \left(\frac{-1}{p}\right) \sum_{y\in\mathbb{F}^*} \omega^0 = \\ &= \left(\frac{-1}{p}\right) (1 + (p-1)) = \left(\frac{-1}{p}\right) p = (-1)^{\frac{p-1}{2}} p. \end{aligned}$$

Applicando la formula (1.8), abbiamo che

$$\tau(1)^{q-1} = (\tau(1)^2)^{\frac{q-1}{2}} = (-1)^{\frac{p-1}{2}\frac{q-1}{2}} p^{\frac{q-1}{2}}$$

e quindi per il criterio di Eulero

$$\tau(1)^{q-1} = (-1)^{\frac{p-1}{2}\frac{q-1}{2}} \left(\frac{p}{q}\right). \tag{1.9}$$

Poiché $\mathbb{E}$ è un campo di caratteristica q, l'elevamento alla potenza q-esima è un omomorfismo di $\mathbb{E}$ in sé. In particolare

$$\tau(1)^q = \left(\sum_{x\in\mathbb{F}^*} \left(\frac{x}{p}\right)\omega^x\right)^q = \sum_{x\in\mathbb{F}^*} \left(\frac{x}{p}\right)^q \omega^{qx} = \sum_{x\in\mathbb{F}^*} \left(\frac{x}{p}\right)\omega^{qx} = \tau(q).$$

Confrontando questa uguaglianza nel campo $\mathbb{E}$ con la (1.9), ed applicando la formula (1.7), si ricava

$$(-1)^{\frac{p-1}{2}\frac{q-1}{2}} \left(\frac{p}{q}\right) = \tau(1)^{q-1} = \tau(q)\tau(1)^{-1} = \left(\frac{q}{p}\right). \tag{1.10}$$

Si tratta questa di un'uguaglianza nel campo $\mathbb{Z}_q$; gli interi in questione pertanto vanno intesi come multipli dell'unità di $\mathbb{Z}_q$. D'altra parte per come sono definiti i simboli di Legendre $\left(\frac{p}{q}\right)$ e $\left(\frac{q}{p}\right)$ appartengono a $\{1,-1\} \subset \mathbb{Z}$, e siccome la caratteristica di $\mathbb{Z}_q$ è il numero dispari q, $1 \neq -1$ in $\mathbb{Z}_q$. Ne segue che da (1.10) deriva l'uguaglianza in $\mathbb{Z}$

$$\left(\frac{p}{q}\right) \cdot \left(\frac{q}{p}\right) = (-1)^{\frac{p-1}{2}\frac{q-1}{2}},$$

come volevasi dimostrare. ■

1.5.3 Il simbolo di Jacobi

Il *simbolo di Jacobi* estende al caso di tutti i numeri dispari il simbolo di Legendre per i numeri primi.

Definizione 7. *Sia n un intero dispari e sia $n = \prod_{i=1}^{l} p_i^{\alpha_i}$ la sua fattorizzazione primaria. Per ogni $m \in \mathbb{Z}$, definiamo il* simbolo di Jacobi *come*

$$\left(\frac{m}{n}\right)_{\mathcal{J}} := \prod_{i=1}^{l} \left(\frac{m}{p_i}\right)^{\alpha_i},$$

dove $\left(\frac{m}{p_i}\right)$ è l'usuale simbolo di Legendre, per ogni $i = 1, \ldots, l$.

Osserviamo che, a differenza del simbolo di Legendre, il simbolo di Jacobi non individua i residui quadratici modulo n. Infatti è possibile che il simbolo di Jacobi calcolato in m sia 1 senza che m sia un R.Q. modulo n. Ad esempio se $n = 9$ ed $m = -1$ abbiamo che

$$\left(\frac{-1}{9}\right)_{\mathcal{J}} = \left(\frac{-1}{3}\right)\left(\frac{-1}{3}\right) = (-1)(-1) = 1$$

mentre -1 non è un R.Q. modulo 9, essendo $(\mathbb{Z}_9)^2 = \{\overline{0}, \overline{1}, \overline{4}, \overline{7}\}$.

Elenchiamo alcune proprietà del simbolo di Jacobi.

Lemma 1.19. *Siano n, n_1, n_2 numeri naturali dispari maggiori di 1, e siano $m, m_1, m_2 \in \mathbb{Z}$. Allora vale quanto segue.*

1. *Se n è primo, $\left(\frac{m}{n}\right)_{\mathcal{J}} = \left(\frac{m}{n}\right)$;*
2. *se $m_1 \equiv m_2 \pmod{n}$, allora $\left(\frac{m_1}{n}\right)_{\mathcal{J}} = \left(\frac{m_2}{n}\right)_{\mathcal{J}}$;*
3. *$\left(\frac{m_1 m_2}{n}\right)_{\mathcal{J}} = \left(\frac{m_1}{n}\right)_{\mathcal{J}}\left(\frac{m_2}{n}\right)_{\mathcal{J}}$;*
4. *$\left(\frac{m}{n_1}\right)_{\mathcal{J}} \cdot \left(\frac{m}{n_2}\right)_{\mathcal{J}} = \left(\frac{m}{n_1 n_2}\right)_{\mathcal{J}}$;*
5. *$\left(\frac{-1}{n}\right)_{\mathcal{J}} = (-1)^{\frac{n-1}{2}}$;*
6. *$\left(\frac{2}{n}\right)_{\mathcal{J}} = (-1)^{\frac{n^2-1}{8}}$.*

Dimostrazione. I punti 1. – 4. si deducono immediatamente dalla definizione di simbolo di Jacobi e dalla moltiplicatività del simbolo di Legendre.

Per i rimanenti punti osserviamo preliminarmente che presi m ed n interi naturali dispari, si ha

$$mn - 1 \equiv (m-1) + (n-1) \pmod{4}$$

e pertanto

$$\frac{mn-1}{2} \equiv \frac{m-1}{2} + \frac{n-1}{2} \pmod{2}.$$

Ne segue che, se $n = \prod_{i=1}^{l} p_i^{\alpha_i}$, per il Lemma 1.15 e la congruenza di sopra,

$$\left(\frac{-1}{n}\right)_{\mathcal{J}} = \prod_{i=1}^{l}\left(\frac{-1}{p_i}\right)^{\alpha_i} = \prod_{i=1}^{l}(-1)^{\frac{p_i-1}{2}\alpha_i} = \prod_{i=1}^{l}(-1)^{\frac{p_i^{\alpha_i}-1}{2}} = (-1)^{\sum_{i=1}^{l}\frac{p_i^{\alpha_i}-1}{2}} = (-1)^{\frac{n-1}{2}},$$

che prova il punto 5. Infine dalla stessa osservazione di sopra segue che se m, n sono interi naturali dispari

$$\frac{m^2n^2-1}{8} \equiv \frac{m^2-1}{8} + \frac{n^2-1}{8} \pmod{2}.$$

Pertanto per la Proposizione 1.16,

$$\left(\frac{2}{n}\right)_{\mathcal{J}} = \prod_{i=1}^{l}\left(\frac{2}{p_i}\right)^{\alpha_i} = \prod_{i=1}^{l}(-1)^{\frac{p_i^2-1}{8}\alpha_i} = \prod_{i=1}^{l}(-1)^{\frac{p_i^{2\alpha_i}-1}{8}} = (-1)^{\sum_{i=1}^{l}\frac{p_i^{2\alpha_i}-1}{8}} = (-1)^{\frac{n^2-1}{8}},$$

provando così 6. ■

Notazione. In virtù del punto 1. del Lemma appena provato, possiamo omettere il pedice $\mathcal{J}$ nella scrittura del simbolo di Jacobi, che verrà pertanto denotato d'ora in poi semplicemente col simbolo $\left(\frac{m}{n}\right)$.

Dal Teorema di reciprocità quadratica di Gauss discende l'analogo risultato per il simbolo di Jacobi.

Teorema 1.20. *(Legge di reciprocità quadratica per il simbolo di Jacobi) Siano m, n due interi positivi dispari. Allora*

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

Dimostrazione. Per prima cosa osserviamo che se m ed n hanno un fattore in comune, allora, per come è definito il simbolo di Legendre, i simboli di Jacobi $\left(\frac{m}{n}\right)$ e $\left(\frac{n}{m}\right)$ sono entrambi nulli. Supponiamo pertanto che m ed n siano coprimi e scriviamo rispettivamente

$$m = \prod_{i=1}^{r} p_i \quad \text{e} \quad n = \prod_{j=1}^{s} q_j$$

come prodotto di primi (non necessariamente distinti). Per convertire $\left(\frac{m}{n}\right) = \prod_{i,j}\left(\frac{p_i}{q_j}\right)$ in $\left(\frac{n}{m}\right) = \prod_{i,j}\left(\frac{q_j}{p_i}\right)$, dobbiamo applicare la Legge di reciprocità quadrata (per i simboli di Legendre) esattamente rs volte. Il numero di fattori (-1) che si generano coincide con il numero di volte che sia p_i che q_j sono congrui a 3 modulo 4, questo è esattamente il prodotto di numeri primi $\equiv 3 \pmod 4$ nelle fattorizzazioni di m ed n. Pertanto $\left(\frac{m}{n}\right) = \left(\frac{n}{m}\right)$, a meno che c'è un numero dispari di primi $\equiv 3 \pmod 4$ in entrambe le fattorizzazioni, nel cui caso si ha $\left(\frac{m}{n}\right) = -\left(\frac{n}{m}\right)$. Ora, un prodotto di primi dispari, come lo sono m ed n, è $\equiv 3 \pmod 4$ se e solo se contiene un numero dispari di primi $\equiv 3 \pmod 4$. Si conclude che $\left(\frac{m}{n}\right) = \left(\frac{n}{m}\right)$, a meno che entrambi m ed n sono $\equiv 3 \pmod 4$, e ciò conclude la dimostrazione. ■

Vediamo ora come può essere utilizzato questo Teorema.
Siano p e q due primi distinti dispari con $p < q$. Allora

$$\left(\frac{p}{q}\right) = (-1)^{\frac{p-1}{2}\frac{q-1}{2}}\left(\frac{q}{p}\right) = (-1)^{\frac{p-1}{2}\frac{q-1}{2}}\left(\frac{r_1}{p}\right),$$

dove r_1 è il resto della divisione di q per p. Scritto $r_1 = 2^{k_1} r_1'$, con r_1' dispari, per il Lemma 1.19, abbiamo che

$$\left(\frac{r_1}{p}\right) = \left(\frac{2^{k_1}}{p}\right)\left(\frac{r_1'}{p}\right) = (-1)^{\frac{p^2-1}{8}k_1}\left(\frac{r_1'}{p}\right).$$

Senza perdere di generalità siamo portati quindi a calcolare $\left(\frac{r_1}{p}\right)$ quando r_1 è dispari. Sfruttiamo ancora il Teorema 1.20 e otteniamo

$$\left(\frac{r_1}{p}\right) = (-1)^{\frac{p-1}{2}\frac{r_1-1}{2}}\left(\frac{p}{r_1}\right) = (-1)^{\frac{p-1}{2}\frac{r_1-1}{2}}\left(\frac{r_2}{r_1}\right),$$

dove r_2 è il resto della divisione di p per r_1. Il procedimento si itera fino all'ultimo resto non nullo, come accade nell'algoritmo di Euclide.

1.6 Pseudoprimi di Eulero

Definizione 8. *Sia n un intero naturale dispari e sia $a \in \mathbb{N}$ coprimo con n. Si dice che n è* pseudoprimo di Eulero rispetto alla base *a se n è composto e*

$$\left(\frac{a}{n}\right) \equiv a^{\frac{n-1}{2}} \quad (mod\ n).$$

A differenza di quanto accade per i numeri che sono pseudoprimi di Fermat, non esiste un equivalente del concetto di numero di Carmichael per i pseudoprimi di Eulero. Detto esplicitamente non esistono interi (dispari) composti che sono pseudoprimi rispetto ad ogni base.

Teorema 1.21. *Sia n un intero naturale dispari e composto. Allora esiste una base $a \in \mathbb{Z}$ per la quale n non è pseudoprimo di Eulero.*

Dimostrazione. Consideriamo dapprima il caso n libero da quadrati.
Sia p un primo che divide n e sia $n = pm$, con ovviamente $(p, m) = 1$. Sia $y + p\mathbb{Z}$ un generatore del gruppo ciclico $\mathbb{Z}_p^*$. Per il Teorema cinese dei resti (Teorema 1.11, esiste una soluzione $a \in \mathbb{Z}$ del seguente sistema di congruenze

$$\begin{cases} x \equiv y \pmod{p} \\ x \equiv 1 \pmod{m} \end{cases}$$

Mostriamo che n non è pseudoprimo di Eulero rispetto alla base a. È banale vedere che $(a, n) = 1$. Inoltre, per le proprietà dei simboli di Jacobi abbiamo che

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p}\right)\left(\frac{a}{m}\right) = \left(\frac{y}{p}\right)\left(\frac{1}{m}\right) = \left(\frac{y}{p}\right) = 1,$$

essendo ovviamente y un non-quadrato modulo p. Ora per valutare $a^{\frac{n-1}{2}}$ modulo n consideriamo l'isomorfismo

$$\begin{aligned} f : \mathbb{Z}_m &\longrightarrow \mathbb{Z}_p \times \mathbb{Z}_m \\ z + n\mathbb{Z} &\longmapsto (z + p\mathbb{Z}, z + m\mathbb{Z}) \end{aligned}$$

Poiché $f(a + n\mathbb{Z}) = (x + p\mathbb{Z}, 1 + m\mathbb{Z})$ abbiamo che

$$f(a^{\frac{n-1}{2}} + n\mathbb{Z}) = (x^{\frac{n-1}{2}} + p\mathbb{Z}, 1^{\frac{n-1}{2}} + m\mathbb{Z}) \neq (-1, -1) = -1_{\mathbb{Z}_p \times \mathbb{Z}_m},$$

e pertanto essendo f un isomorfismo $a^{\frac{n-1}{2}} + n\mathbb{Z} \neq -1 + n\mathbb{Z}$, cioè $a^{\frac{n-1}{2}} \not\equiv -1 \pmod{n}$.
Trattiamo ora il caso n non libero da quadrati.
Sia $n = p^k m$, con p primo, $(p, m) = 1$ e $k \geq 2$ e mostriamo che n non è pseudoprimo di Eulero rispetto alla base $a = 1 + l$ con $l = p^{k-1}m$. Ogni primo p_i che divide n divide necessariamente l, pertanto a è congruo ad 1 modulo p_i. In particolare $(a, n) = 1$ e

$$\left(\frac{a}{n}\right) = \prod_i \left(\frac{a}{p_i}\right)^{\alpha_i} = \prod_i 1 = 1.$$

Se n fosse pseudoprimo di Eulero rispetto ad a allora dovremmo avere che $a^{\frac{n-1}{2}} \equiv 1 \pmod{n}$. In particolare $a^{\frac{n-1}{2}} \equiv 1 \pmod{p^k}$. Ma posto $u = \frac{n-1}{2}$ abbiamo che

$$a^u - 1 = (1 + p^{k-1}m)^u - 1 = \sum_{j=1}^{u} \binom{u}{j} (p^{k-1}m)^j \equiv up^{k-1}m \pmod{p^k}$$

ed essendo m coprimo con p affiché $a^u \equiv 1 \pmod{p^k}$ si deve avere che $p|u$. Ma ciò è assurdo poiché $u = \frac{n-1}{2}$ e p divide n. ■

Corollario 1.22. *Sia n un intero dispari. Allora n è un numero primo se e solo se n è pseudoprimo di Eulero rispetto ad ogni base a, $(a, n) = 1$.*

Capitolo 2

Complessità ed algoritmi

In questo capitolo presentiamo alcune nozioni basilari di Teoria della Complessità. L'intento è quello di introdurre un sistema per valutare i protocolli crittografici che esporremo nei capitoli successivi. Calcoleremo pertanto la complessità dei principali algoritmi per l'aritmetica classica e modulare, per poi analizzare l'algoritmo di Euclide esteso e le procedure di estrazioni di radici quadrate su $\mathbb{N}$ e su campi finiti.

2.1 Cenni di teoria della complessità computazionale

Per misurare l'efficienza di un algoritmo in modo univoco, bisogna definire una metrica che sia indipendente dalle tecnologie utilizzate. Il modello di calcolo generico che viene adottato come riferimento standard è la cosiddetta *macchina di Turing*. Ogni modello di calcolo scelto (dalla macchina RAM, ai computer reali), ai fini della classificazione dei problemi, si comporta esattamente come una macchina di Turing. Nella teoria della calcolabilità vale infatti la seguente ipotesi.

Tesi di Church-Turing. *La classe delle funzioni calcolabili coincide con quella delle funzioni calcolabili da una macchina di Turing.*

Nel 1936 Alan Turing, per risolvere il problema di decisione (*Entscheidungsproblem*) proposto da David Hilbert, descrisse per la prima volta quella che sarebbe stata poi definita macchina di Turing ([26]). Senza entrare nei dettagli, che ci allontanerebbero dallo scopo di questo corso, possiamo considerare la macchina di Turing come una macchina ideale in grado di manipolare i dati contenuti su un nastro di lunghezza potenzialmente infinita, secondo un prefissato sistema di regole ben definite. Detto altrimenti, è un modello astratto che definisce una macchina in grado di eseguire algoritmi e dotata di un nastro potenzialmente infinito su cui può leggere e scrivere dei simboli. Per una

definizione rigorosa delle macchine di Turing, la loro analisi, ed in generale, per una conoscenza più approfondita circa i teoremi fondamentali della calcolabilità e della teoria della complessità, rimandiamo ai corsi di Logica e ai libri [13] e [22]. Di seguito ci limiteremo a brevi considerazioni, fondamentali per il nostro proseguo.

Si è soliti organizzare i problemi in base alla loro *classe di complessità*. Elenchiamo quelle che sono le tre classi più significative per i nostri scopi.

- Un problema appartiene alla classe P (equivalentemente è *risolvibile in tempo polinomiale*) se esiste una macchina di Turing che è in grado di risolvere un'istanza del problema di ampiezza n in al più $p(n)$ passi, per qualche polinomio $p(x) \in \mathbb{R}[x]$.
- Un problema appartiene alla classe NP (equivalentemente è *risolvibile in tempo polinomiale non deterministico*) se esiste una macchina di Turing che può verificare la correttezza di una soluzione proposta, ad un'istanza di ampiezza n del problema, in al più $p(n)$ passi, per qualche $p(x) \in \mathbb{R}[x]$.
- Un problema appartiene alla classe $\mathsf{ExpTime}$ (equivalentemente è *risolvibile in tempo esponenziale*) se esiste una macchina di Turing che è in grado di risolvere un'istanza del problema di ampiezza n in al più $2^{p(n)}$ passi, per qualche $p(x) \in \mathbb{R}[x]$.

Ovviamente un problema di complessità maggiore è più difficile da risolvere, anche proprio da un punto di vista pratico. Ad esempio, immaginiamo che per risolvere un problema occorrano n^3 passi (dove n è l'ampiezza del problema). Allora, nell'ipotesi che ogni passo impieghi un nanosecondo, se $n = 1.000$ risolviamo il problema in un secondo, mentre se $n = 10.000$ in tre mesi. Se invece occorrono 2^n passi per risolvere lo stesso problema, allora per avere la soluzione già solo per $n = 100$, dovremmo aspettare più dell'età dell'universo! La regola generale è pertanto che i problemi risolvibili in tempo polinomiale siano facili, mentre quelli risolvibili in tempo esponenziale siano difficili.

Teorema 2.1. $\mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{ExpTime}$.

Poiché questo non è un corso sulla complessità, non proveremo in dettaglio il Teorema. Limitiamoci ad alcuni commenti che aiutano a chiarire meglio i concetti.
La prima inclusione vale in quanto se si è in grado di trovare una soluzione in tempo polinomiale, allora sempre in tempo polinomiale è possibile verificare l'esattezza di una eventuale soluzione proposta. La seconda inclusione invece risulta vera in quanto, se possiamo verificare una soluzione proposta in tempo polinomiale, allora tale verifica userà un numero polinomiale, $p(n)$, di quadri della macchina di Turing. In particolare, se q indica la cardinalità dell'alfabeto, allora i $p(n)$ quadri possono produrre al più $q^{p(n)}$ possibili stringhe.

Enfatizziamo il fatto che è relativamente facile mostrare che un problema appartiene ad una particolare classe di complessità. Per procedere in modo rigoroso, dobbiamo mostrare l'esistenza di una macchina di Turing efficiente; in pratica però, è sufficiente trovare un algoritmo che risolva il problema in modo efficiente. Tradurre questo algoritmo in una macchina di Turing può certamente aumentare il numero di passi, ma mai abbastanza da alterare le conclusioni cui si è giunti. La cosa veramente complicata è mostrare che un certo problema *non* appartiene a una particolare classe di complessità, poiché dovremmo mostrare che nessuna macchina di Turing, o nessun algoritmo, è in grado di risolvere il dato problema in modo efficiente.
È ben noto che la classe P è strettamente contenuta in $\mathsf{ExpTime}$, pertanto almeno una delle inclusioni del Teorema 2.1 deve essere propria. È ormai celebre la seguente

Congettura. $\mathsf{P} \subsetneq \mathsf{NP}$.

Questo è annoverato come uno dei sette problemi del millennio e il Clay Mathematical Institute di Providence (USA) offre ben 1.000.000 $ per una prova o una confutazione di tale congettura.
Una sottoclasse particolare di NP è costituita dai cosiddetti problemi NP-*completi*. Ne sono esempi il problema del commesso viaggiatore[1] e il problema del matrimonio a tre sessi[2]. In generale, i problemi NP-completi sono i più difficili fra quelli NP. Questi sono stati a lungo studiati, e per nessuno è stato ancora trovato un algoritmo che possa risolverlo in tempo polinomiale. Se un tale algoritmo dovesse un giorno essere scoperto per un problema NP-completo, allora automaticamente si avrebbe che $\mathsf{P} = \mathsf{NP}$ e la congettura sarebbe così confutata.

2.2 Complessità di alcune operazioni elementari

La complessità dei protocolli crittografici si misura contando il numero di *operazioni elementari* (o *bit-operations*) che un computer deve svolgere per eseguirli. Nel far ciò, è consuetudine rappresentare i numeri interi in base due, poiché un computer legge i dati sotto forma di stringhe di bit.

[1] Data una rete di città, connesse tramite delle strade, trovare il percorso di minore lunghezza che un commesso viaggiatore deve seguire per visitare tutte le città una e una sola volta per poi tornare alla città di partenza.

[2] Dati tre insiemi finiti e disgiunti X, Y e Z della stessa cardinalità n e dato $A \subseteq X \times Y \times Z$, determinare un algoritmo per decidere se esiste $B \subseteq A$, di cardinalità n, per cui per ogni $(x, y, z) \neq (x', y', z') \in B$ si ha $x \neq x'$, $y \neq y'$ e $z \neq z'$.

Dato un arbitrario intero positivo n, sia

$$n = \sum_{i=0}^{k} a_i 2^i,$$

la sua rappresentazione in base due (ovvero dove ogni $a_i \in \{0, 1\}$ e $a_k = 1$). Chiamiamo *lunghezza binaria* di n il numero di bit necessario per rappresentare n in base due, e indichiamo tale numero con $\text{size}\,(n)$. In questo caso, $\text{size}\,(n) = k + 1$. È immediato osservare che

$$2^k \leq n \leq \sum_{i=0}^{k} 2^i = 2^{k+1} - 1 < 2^{k+1}.$$

In particolare, la $\text{size}\,(n)$ altro non è che la parte intera di $\log_2(n)$ più uno, in simboli

$$\text{size}\,(n) = k + 1 = \lfloor \log_2(n) \rfloor + 1 = \left\lfloor \frac{\log(n)}{\log(2)} \right\rfloor + 1 \simeq \lfloor 1,4427 \log(n) \rfloor + 1$$

(dove il simbolo log indica il logaritmo naturale $\log_e$). Una sovrastima per $\text{size}\,(n)$ è pertanto fornita da $\lfloor \log_2(n) \rfloor + 1$.

Per stimare i tempi di esecuzione e quindi la complessità dei vari algoritmi, si fa uso della notazione *O-grande*.

Definizione 9. *Date due funzioni, f e g, di dominio $\mathbb{N}$ e codominio $\mathbb{R}^+$, si dice che f* è un O-grande di *g, e si scrive $f \in O(g)$, se esistono $n_0 \in \mathbb{N}$ e $c \in \mathbb{R}^+$, tali che*

$$f(n) \leq cg(n) \quad \forall n \geq n_0.$$

In pratica, $f \in O(g)$ se e solo se la funzione f cresce definitivamente meno di un multiplo scalare di g. Si osservi che $f \in O(1)$ se e solo se f è limitata da una costante.

Possiamo pertanto ora avvalerci della seguente definizione, più pratica, per definire la classe P.

Definizione 10. *Un problema si dice* di tipo polinomiale *(ovvero appartiene alla classe P) se esiste un algoritmo A tale che, detta f la funzione che conta il numero di operazioni elementari svolte da A in funzione del dato iniziale n, esiste un intero $m \in \mathbb{N}$ tale che $f \in O\big(\log_2^m n\big) = O\big(\text{size}\,(n)^m\big)$.*

Analizziamo ora quanto l'aritmetica di base è dispendiosa, in termini di operazioni elementari (in breve o.e.).

Somma.
Sommiamo due interi a ed b entrambi di k cifre binarie (se uno dei due ha un numero di cifre inferiore dell'altro, possiamo sempre pensare di aggiungere degli zero a sinistra in modo da avere entrambi i numeri della stessa lunghezza in termini di bit). Detti $a = \sum_{i=0}^{k-1} a_i 2^i$ e $b = \sum_{i=0}^{k-1} b_i 2^i$, calcoliamo la somma $a + b = \sum_{i=0}^{k} c_i 2^i$ utilizzando il procedimento che impariamo alle scuole elementari, ovvero il seguente pseudo-codice.

1. per $j = 0$ poni $r = 0$
2. per $j = 0, 1, \ldots, k-1$ poni
   (a) $u := a_j + b_j + r$
   (b) $c_j := u \pmod 2$
   (c) $r := \lfloor \frac{u}{2} \rfloor$
   (d) $j = j + 1$

L'intero r indica ad ogni passo il riporto della somma del passo precedente.
Notiamo che ad ogni passo ci sono due operazioni elementari. Pertanto nel caso in cui gli interi hanno al massimo k cifre binarie, si ha un totale di al più $2k$ o.e. In altri termini, per sommare due interi, a e b, di al più n cifre decimali (cioè $\lfloor \log_2(n) \rfloor$ cifre binarie), occorrono al più $2\lfloor \log_2(n) \rfloor$ o.e. L'operazione di somma è pertanto un $O\big(\log_2(n)\big)$, dove n è il numero di cifre decimali del massimo dei due numeri. In altri termini, sommare a e b richiede un tempo polinomiale dell'ordine $O\big(\max\{\text{size}(a), \text{size}(b)\}\big)$.

Esercizio 2.1. Mostrare per esercizio che la sottrazione richiede un numero di o.e. che è lo stesso della somma. (Suggerimento: utilizzare il complemento bit-a-bit, oppure calcolare prima il segno e lavorare con gli interi in valore assoluto).

Prodotto.
Consideriamo ora l'operazione di prodotto tra due interi.
Se a consiste di k_1 cifre binarie e b di $k_2 \leq k_1$ cifre binarie, allora per completare l'operazione di prodotto $a \cdot b$, secondo la procedura che impariamo alle scuole elementari, dobbiamo portare a termine k_2 somme di interi che hanno al più $k_1 + k_2 \leq 2k_1$ cifre binarie. Poiché, per quanto abbiamo appena visto, ogni somma richiede al massimo $2(2k_1)$ o.e., si ha che, al massimo, si eseguiranno $4k_1^2$ o.e.. Dunque la complessità di questo algoritmo per il prodotto di due interi è un $O(k^2)$, dove $k = \max\{\text{size}(|a|), \text{size}(|b|)\}$ (per gli interi negativi, si può prima calcolare il segno del prodotto, cosa computazionalmente facile, poi effettuare il prodotto dei due interi in modulo).

Esistono algoritmi per il prodotto di due interi che richiedono un numero di o.e. inferiore a $O(k^2)$. Uno di questi è dovuto a Anatolii Karatsuba (si veda [17] o [19]).
Siano a e b due interi arbitrari e sia $k+1 = \max\{\lfloor \log_2(a) \rfloor, \lfloor \log_2(b) \rfloor\}$. Eventualmente aggiungendo zeri, possiamo supporre che a e b abbiano lo stesso numero di bit, e siano

$$a = \sum_{i=0}^{k} a_i 2^i \quad \text{e} \quad b = \sum_{i=0}^{k} b_i 2^i.$$

Indichiamo anche con $T(k)$ il costo dell'operazione di moltiplicazione di due interi aventi k cifre binarie. Nel seguito possiamo anche limitarci a supporre k pari. Scriviamo a e b ciascuno come combinazione lineare di due interi lunghi al massimo $k/2$ bit, in questo modo

$$a = \sum_{i=0}^{k} a_i 2^i = \sum_{i=0}^{\frac{k}{2}-1} a_i 2^i + 2^{\frac{k}{2}} \cdot \sum_{i=\frac{k}{2}}^{k} a_i 2^{i-\frac{k}{2}} = u_0 + 2^{\frac{k}{2}} u_1,$$

$$b = \sum_{i=0}^{k} b_i 2^i = \sum_{i=0}^{\frac{k}{2}-1} b_i 2^i + 2^{\frac{k}{2}} \cdot \sum_{i=\frac{k}{2}}^{k} b_i 2^{i-\frac{k}{2}} = v_0 + 2^{\frac{k}{2}} v_1.$$

Abbiamo che

$$ab = u_0 v_0 + 2^{\frac{k}{2}}(u_0 v_1 + u_1 v_0) + u_1 v_1 2^{k-1}.$$

Fare queste tre somme non è complicato, in quanto in genere gli addendi sono di ordini differenti. Il problema è quindi stimare il calcolo di $u_0 v_0$, $u_0 v_1 + u_1 v_0$ e $u_1 v_1$. Possiamo scrivere

$$u_0 v_1 + u_1 v_0 = -(u_0 - u_1)(v_0 - v_1) + u_0 v_0 + u_1 v_1,$$

da cui segue la formula

$$T(k) = 3T\left(\frac{k}{2}\right) + O(k).$$

Possiamo porci nel caso $k = 2^h$ che ci consente di trovare le seguenti maggiorazioni alla stima

$$T(k) \leq 3T\left(\frac{k}{2}\right) + O(k) \leq 3^2 T\left(\frac{k}{2^2}\right) + 2O(k) \leq \ldots \leq 3^h T(1) + hO(k).$$

Ora $T(1)$ rappresenta il numero di o.e. per moltiplicare due interi lunghi al massimo un bit, e pertanto $T(1) = 1$. Inoltre $h = \log_2(k) = \frac{\log_3(k)}{\log_3(2)} = \log_3(k)\log_2(3)$, segue perciò che

$$\begin{aligned} T(k) &\leq (3^{\log_3(k)})^{\log_2(3)} + \log_2(k) O(k) \\ &= O(k^{\log_2(3)}) + O(k \log_2(k)) = O(k^{\log_2(3)}), \end{aligned}$$

essendo $k\log_2(k) \in O(k^{\log_2(3)})$. Da notare che $\log_2(3) \sim 1,58 < 2$ e che quindi questo algoritmo migliora il precedente.
Altri algoritmi, che usano le trasformate di Fourier, migliorano ulteriormente questa stima. In ogni caso al di là delle possibili migliorie, come la somma anche il prodotto richiede un tempo polinomiale per essere effettuato da una macchina.
Divisione con resto.
Provare che la complessità della divisione con resto di a per b ($a \geq b$) richiede $O\big(\text{size}\,(b)\,\text{size}\,(q)\,\big)$ o.e., dove q è il quoziente della divisione. In particolare, il costo della divisione è polinomiale ed in prima istanza può essere considerato un $O\big(\text{size}\,(a)^2\,\big)$.

2.3 L'algoritmo di Euclide

L'algoritmo di Euclide permette di determinare il massimo comune divisore, (a, b), di due numeri interi, a e b, in modo molto efficiente. Si basa su questo risultato (per la cui dimostrazione rimandiamo a [9]).

Teorema 2.2. *Siano a e b due interi. Se $b = 0$, allora $(a, b) = |a|$. Se $b \neq 0$, allora*

$$(a, b) = (|b|\,, a\,(\text{mod}\,|b|)),$$

dove $a\,(\text{mod}\,|b|)$ indica il resto della divisione di a con il modulo $|b|$).

L'algoritmo di Euclide procede secondo questo pseudocodice.

```
int Euclide(int a, int b, int gcd) // prototipo della funzione Euclide //
begin
    int r;
    a=|a|
    b=|b|
    while(b != 0)        // ciclo di iterazione
        r = a % b;       // operazione modulo
        a = b;
        b = r;
    end while
    gcd = a
end
```

(Spiegazione. Dapprima l'algoritmo rimpiazza a e b rispettivamente con i loro valori assoluti. Finché b è diverso da zero, l'algoritmo scambia a con b e b con $a \pmod b$. Quando $b = 0$ l'algoritmo termina e fornisce in output il valore di a).

Si tratta di un algoritmo molto efficiente, il che è fondamentale per le sue applicazioni nei protocolli crittografici. Calcoliamo il numero di iterazioni necessarie. Senza ledere la genericità, possiamo assumere $a > b > 0$. Indichiamo con $\{r_i\}_i$ la sequenza dei resti delle divisioni successive e con $\{q_i\}_i$ quella dei quozienti; per la precisione, siano

$$\begin{cases} r_0 = a \\ r_1 = b \\ r_{i+2} = r_i - r_{i+1}q_{i+1} \quad \text{con } 0 \leq r_{i+2} < r_{i+1} \end{cases} \qquad \text{e} \qquad q_{i+1} = \left\lfloor \frac{r_i}{r_{i+1}} \right\rfloor$$

Al passo $(i+2)$-esimo si esegue la divisione con resto di r_i per r_{r+1}. Poiché così facendo si trova una successione strettamente decrescente di resti non negativi, ad un certo punto esisterà un intero positivo m per cui $r_{m+1} = 0$, il valore di (a, b) sarà pertanto r_m. Preoccupiamoci intanto di stimare il numero m di iterazioni necessarie.

Lemma 2.3. *$q_i \geq 1$, per ogni $i = 1, \ldots, m-1$ e $q_m \geq 2$.*

Dimostrazione. Poiché $r_{i-1} > r_i > r_{i+1}$, segue che $q_i \geq 1$. Se fosse $q_m = 1$, allora $r_{m-1} = r_m$, il che è impossibile. ■

Proposizione 2.4. *Siano a e b due interi positivi, con $a \geq b$. Sia $\theta = (1 + \sqrt{5})/2$. Se m indica il numero di iterazioni che l'algoritmo di Euclide impiega per calcolare (a, b), allora*

$$m \leq \frac{\log_2(b)}{\log_2(\theta)} + 1 < 1,441 \log_2(b) + 1.$$

In particolare $m \in O(\text{size}\,(b))$.

Dimostrazione. Per l'Esercizio 2.2, possiamo assumere che a e b siano coprimi, dunque $(a, b) = r_m = 1$. Proviamo per induzione su $j = 0, 1, \ldots, m$ che

$$r_{m-j} \geq \theta^j.$$

Se $j = 0$, allora $r_m = 1 = (a, b)$ e dunque $r_m \geq \theta^0 = 1$.
Supponiamo che valga $r_{m-j} \geq \theta^j$ e stimiamo $r_{m-(j+1)}$. Si ha che

$$r_{m-j-1} = r_{m-j}q_{m-j} + r_{m-(j-1)} \geq r_{m-j} + r_{m-(j-1)} \geq \theta^j + \theta^{j-1} = \theta^j(1 + 1/\theta) = \theta^{j+1},$$

dove si è usato il fatto che θ è radice del polinomio $x^2 - x - 1$.
In particolare per $j = m - 1$ otteniamo che $b = r_1 \geq \theta^{m-1}$, da cui segue facilmente la tesi. ■

Esercizio 2.2. Provare che il numero di iterazioni che l'algoritmo di Euclide impiega a calcolare $(a, b) = d$ coincide con quello che impiega a stabilire che $(a/d, b/d) = 1$.

2.3.1 Algoritmo di Euclide esteso

Il massimo comune divisore tra due interi, a e b, è il più piccolo intero positivo che si scrive come combinazione lineare a coefficienti interi di a e b,

$$(a, b) = xa + yb, \qquad \text{con } x, y \in \mathbb{Z}.$$

Supponendo come prima che $a > b > 0$, il metodo per trovare la combinazione lineare suddetta consiste nel costruire le due successioni di interi $\{x_i\}_{i=0}^m$ ed $\{y_i\}_{i=0}^m$ così definite

$$\begin{cases} x_0 = 1 \\ x_1 = 0 \\ x_{i+1} = x_i q_i + x_{i-1},\ i = 1, \dots, m. \end{cases} \quad \text{e} \quad \begin{cases} y_0 = 0 \\ y_1 = 1 \\ y_{i+1} = y_i q_i + y_{i-1},\ i = 1, \dots, m. \end{cases} \tag{2.1}$$

Lasciamo al lettore provare per induzione la seguente

Proposizione 2.5. *1. Le successioni $\{x_i\}_i$ ed $\{y_i\}_i$ sono crescenti.*

2. Per ogni $i = 0, \dots, m+1$ vale $r_i = (-1)^i x_i a + (-1)^{i+1} y_i b$.

In particolare abbiamo che

$$(a, b) = r_m = xa + yb \tag{2.2}$$

dove $x = (-1)^m x_m$ e $y = (-1)^{m+1} y_m$.

2.3.2 Analisi dei costi computazionali

Vediamo ora quanto è dispendioso l'intero algoritmo di Euclide esteso.
Incominciamo col trovare una stima per la grandezza dei combinatori x ed y che compaiono nella formula (2.2). Definiamo le seguenti matrici 2×2:

$$E_i = \begin{pmatrix} q_i & 1 \\ 1 & 0 \end{pmatrix},\ 1 \le i \le m, \qquad \text{e} \qquad T_i = \begin{pmatrix} y_i & y_{i-1} \\ x_i & x_{i-1} \end{pmatrix},\ 1 \le i \le m+1.$$

Abbiamo che la coppia di sistemi (2.1) è equivalente alla seguente uguaglianza fra matrici

$$T_{i+1} = T_i E_i, \quad \forall 1 \le i \le m. \tag{2.3}$$

Essendo T_1 la matrice identica, otteniamo

$$T_{m+1} = E_1 E_2 \dots E_m. \tag{2.4}$$

Poniamo ora $S_i = E_{i+1} E_{i+2} \dots E_m$, per ogni $i = 0, 1, \dots, m-1$. Si noti che $S_0 = T_{m+1}$ ed $S_{m-1} = E_m$. Poniamo

$$S_i = \begin{pmatrix} u_i & v_i \\ u_{i+1} & v_{i+1} \end{pmatrix}, \quad \forall\, i = 0, 1, \dots, m.$$

Dalla ricorsione $S_{i-1} = E_i S_i$, si ottiene per ogni $i = 1, \ldots, m$

$$\begin{cases} u_{i-1} = q_i u_i + u_{i+1} \\ v_{i-1} = q_i v_i + v_{i+1} \end{cases} \tag{2.5}$$

Lemma 2.6. *1. Per ogni $i = 0, \ldots, m$, si ha che $0 \leq v_i \leq \frac{r_i}{2(a,b)}$.*

2. Per ogni $i = 1, \ldots, m$, si ha che $x_i \leq \frac{b}{2(a,b)}$ e $y_i \leq \frac{a}{2(a,b)}$.
In particolare, $x_i, y_i \in O(a)$.

Dimostrazione. 1. Facciamo induzione su $j = m - i$, ovvero induzione a ritroso su i. Se $i = m$ dal fatto che $S_{m-1} = E_m$ segue che $v_m = 0$ e quindi $v_m < r_m/(2(a,b))$. Sia ora $i < m$ e supponiamo il risultato vero per ogni $j \geq i$; proviamolo per $i - 1$. Dalla (2.5) abbiamo che

$$v_{i-1} = q_i v_i + v_{i+1} \leq (q_i r_i + r_{i+1})/(2(a,b)) = r_{i-1}/(2(a,b)).$$

2. Dal fatto che $S_0 = T_{m+1}$ segue che $x_m = v_1$ e $y_m = v_0$. Pertanto, per il punto 1., otteniamo la stima voluta per $i = m$. Poiché le successioni $\{x_i\}_i$ e $\{y_i\}_i$ sono crescenti per il Lemma 2.5, la tesi è vera per ogni $i = 1, \ldots, m$. ■

Siamo ora in grado di dimostrare che il costo computazionale dell'algoritmo di Euclide è sorprendentemente basso, per intenderci dello stesso ordine di grandezza della semplice moltiplicazione fra due interi.

Teorema 2.7. *L'algoritmo di Euclide (esteso) usa un tempo $O(\text{size}\,(a)\,\text{size}\,(b))$ sia per calcolare (a,b) che una sua rappresentazione come $(a,b) = xa + yb$, con x, y in $\mathbb{Z}$.*

Dimostrazione. Se $b = 0$, l'algoritmo impiega un solo passo per calcolare $(a,b) = |a|$. Il tempo per questa operazione è senz'altro un $O(\text{size}\,(a)\,1) = O(\text{size}\,(a)\,\text{size}\,(b))$. Assumiamo pertanto $a > b > 0$. L'algoritmo calcola i resti $\{r_i\}_{i=0}^{m+1}$ e i quozienti $\{q_i\}_{i=1}^{m}$. Poiché r_{i+1} è il resto della divisione di r_{i-1} con r_i (per $1 \leq i \leq m$), il costo per fare questa operazione richiede esattamente $O(\text{size}\,(r_i)\,\text{size}\,(q_i))$ o.e.. Ora, per ogni $i \geq 1$ si ha che $r_i \leq b$; inoltre, $\text{size}\,(q_i) = \log_2(q_i) + 1$, per ogni $1 \leq i \leq m$. Ne segue che il tempo impiegato dall'algoritmo euclideo (semplice) è

$$T_{\text{Euclid}}(a,b) = O\left(\text{size}\,(b)\left(\sum_{i=1}^{m} \text{size}\,(q_i)\right)\right) = O\left(\text{size}\,(b)\left(m + \sum_{i=1}^{m} \log_2(q_i)\right)\right). \tag{2.6}$$

Dalla Proposizione 2.4, si ha poi che

$$m = O(\text{size}\,(b)). \tag{2.7}$$

Inoltre,

$$\begin{aligned} a = r_0 &= q_1 r_1 + r_2 \geq q_1 r_1 = q_1(q_2 r_2 + r_3) \\ &\geq q_1 q_2 r_2 \geq \ldots \geq q_1 q_2 \ldots q_m, \end{aligned}$$

e pertanto

$$\sum_{i=1}^{m} \log_2(q_i) = \log_2\left(\prod_{i=1}^{m} q_i\right) = O(\text{size}\,(a)). \tag{2.8}$$

Dalle formule (2.6), (2.7) e (2.8) segue che il costo computazionale dell'algoritmo euclideo semplice è dell'ordine di $O(\text{size}\,(a)\,\text{size}\,(b))$.

Stimiamo ora il tempo necessario che l'algoritmo esteso impiega per calcolare i coefficienti x ed y. Al primo passo, abbiamo

$$x_2 = q_1 x_1 + x_0 = 1, \quad y_2 = q_1 y_1 + y_0 = q_1.$$

Questo conto è sicuramente un $O(\text{size}\,(q_1)) = O(\text{size}\,(a))$. Poi vengono calcolati per ogni $2 \leq i \leq m$ gli elementi

$$x_{i+1} = q_i x_i + x_{i-1} \quad \text{e} \quad y_{i+1} = q_i y_i + y_{i-1}.$$

Per il Lemma 2.6(2), abbiamo che $x_i, y_i = O(a)$, per $0 \leq i \leq m$. Il tempo necessario per il calcolo di x e y è pertanto

$$\begin{aligned} T_{\text{ExtendedEuclid}}(a,b) &= O\left(\text{size}\,(a)\left(1 + \sum_{i=2}^{m} \text{size}\,(q_i)\right)\right) \\ &= O\left(\text{size}\,(a)\left(m + \sum_{i=2}^{m} \log_2(q_i)\right)\right). \end{aligned}$$

Come sopra, è immediato vedere che $\prod_{i=2}^{m} q_i \leq b$, da cui segue facilmente la tesi. □

Esercizio 2.3. Si dimostri la Proposizione 2.5.

Esercizio 2.4. Con le stesse notazioni si sopra, provare che $|x| \leq \frac{b}{2(a,b)}$ e $|y| \leq \frac{a}{2(a,b)}$.

2.4 Estrazioni di radici quadrate in $\mathbb{N}$.

Vediamo ora un algoritmo che assegnato un intero naturale positivo n calcola il valore di $x = \lfloor \sqrt{n} \rfloor$. La strategia consiste nel rappresentare n in base 2 e ad ogni passo trovare una cifra binaria per x. Sia dunque

$$n = \sum_{i=0}^{k} n_i 2^i,$$

con $n_i \in \{0,1\}$ e $n_k = 1$, dove $k = \text{size}(n) - 1$. È immediato vedere che $\lfloor\sqrt{n}\rfloor$ ha esattamente $\lfloor k/2\rfloor + 1$ cifre binarie. Infatti, poiché $2^k \leq n < 2^{k+1}$, si ha che $2^{k/2} \leq \sqrt{n} < 2^{(k+1)/2}$. D'altra parte, $\lfloor k/2\rfloor \leq k/2$ e $(k+1)/2 \leq \lfloor k/2\rfloor + 1$ e pertanto

$$2^{\lfloor k/2\rfloor} \leq 2^{k/2} \leq \sqrt{n} < 2^{(k+1)/2} \leq 2^{\lfloor k/2\rfloor+1},$$

cioè $\text{size}(\sqrt{n}) = \lfloor k/2\rfloor + 1$. Scriviamo pertanto

$$\lfloor\sqrt{n}\rfloor = x = \sum_{i=0}^{h} x_i 2^i$$

con $h = \lfloor k/2\rfloor$ e determiniamo ricorsivamente le cifre $x_i \in \{0,1\}$. Ovviamente poiché $2^h \leq x < 2^{h+1}$, si ha $x_h = 1$.
Osserviamo che posto $\sigma_{h-1} := 2^h + 2^{h-1}$ vale che

$$\sigma_{h-1} \leq \lfloor\sqrt{n}\rfloor \Longleftrightarrow x_{k-1} = 1.$$

Inoltre,

$$\sigma_{h-1}^2 \leq n \Longleftrightarrow \sigma_{h-1} \leq \sqrt{n} \Longleftrightarrow \sigma_{h-1} \leq \lfloor\sqrt{n}\rfloor.$$

Ne segue che $x_{h-1} = 1$ se e solo se $\sigma_{h-1}^2 \leq n$; tale confronto si può fare in maniera agevola, basta elevare al quadrato $\sigma_{h-1} = 2^h + 2^{h-1}$.
Supponiamo ora di aver trovato le cifre $x_{h-1}, x_{h-3}, \ldots, x_{j+1}$, vediamo come determinare x_j. Definiamo

$$\sigma_j := 2^j + \sum_{i=j+1}^{h} x_i 2^i.$$

Come prima abbiamo che

$$\sigma_j^2 \leq n \Longleftrightarrow \sigma_j \leq \sqrt{n} \Longleftrightarrow \sigma_j \leq \lfloor\sqrt{n}\rfloor \Longleftrightarrow x_j = 1.$$

Dal confronto di σ_j^2 con n siamo in grado di determinare con esattezza x_j. Questo ciclo arriva fino a $j = 0$, dopodiché x è completamente determinato.
Stimiamo il costo dell'intero ciclo in termini di o.e.. Ogni passo consiste sostanzialmente nell'elevare al quadrato l'elemento σ_j e confrontarlo con n. Ora σ_j ha un $O(\log_2 n)$ cifre binarie, quindi ogni passo costa al più $O(\log_2^2 n)$ o.e.. In tutto abbiamo $O(\frac{1}{2}\log_2 n) = O(\log_2 n)$ cifre binarie, quindi il numero di passi da eseguire è al più $O(\log_2 n)$. Ne risulta che il costo dell'intero algoritmo è un $O(\log_2^3 n)$.

2.5 Complessità dell'aritmetica modulare di base.

Molti protocolli crittografici sfruttano l'aritmetica degli anelli finiti, ad esempio di $\mathbb{Z}_m$, o dei campi di Galois $\mathbb{F}_q$ d'ordine q potenza di primo. In questa breve sezione, formalizziamo alcune operazioni di base della cosiddetta aritmetica modulare, e andiamo a calcolare i relativi costi computazionali.

Potenze.

Sia m un intero positivo e consideriamo l'anello $\mathbb{Z}_m$. Esporremo un algoritmo per calcolare, in modo piuttosto efficiente, le potenze degli elementi di $\mathbb{Z}_m$. Questa è una situazione che ricorre assai spesso (ad esempio come si è visto nel capitolo precedente, se $m = p$ e vogliamo determinare il simbolo di Legendre per un certo $a \in \mathbb{Z}$ dobbiamo eseguire la potenza $(a + p\mathbb{Z})^{(p-1)/2}$). Osserviamo che calcolare la potenza $x^{(m-1)/2}$ per un certo $x \in \mathbb{Z}_m$ può essere assai dispendioso se utilizziamo un algoritmo banale. Infatti, se rappresentiamo ogni elemento di $\mathbb{Z}_m$ come un intero di al più $\lfloor \log_2(m) \rfloor + 1$ cifre binarie, allora il calcolo di x^2 richiederebbe circa $\log_2^2(m)$ o.e., per fare poi $x^3 = x^2 \cdot x$ dovremmo eseguire circa $\log_2^2(m) + \log_2^2(m) = 2\log_2^2(m)$ o.e., e così via fino ad ottenere $x^{(m-1)/2}$ dopo circa $\frac{m-1}{2}\log_2^2(m)$ o.e., numero che di per sé potrebbe essere intrattabile. È opportuno pertanto trovare un algoritmo più agile per fare le potenze di elementi di $\mathbb{Z}_m$. Quello che presentiamo prende il nome di *Fast Powering Algorithm.*

Siano $x \in \mathbb{Z}_m$ ed $n \in \mathbb{N}$, vediamo come fare per calcolare x^n in modo efficiente. Il primo passo consiste nel rappresentare n in base due. Sia pertanto $n = \sum_{i=0}^{k} n_i 2^i$, con $a_i \in \{0, 1\}$ ed $a_k = 1$. Abbiamo che

$$x^n = \prod_{i=0}^{k} (x^{2^i})^{n_i}.$$

Definiamo ora per ogni $i = 0, 1, \ldots, k-1$,

$$\begin{cases} s_0 = x^{2^0} = x \\ s_{i+1} = x^{2^{i+1}}. \end{cases}$$

Si noti che $s_{i+1} = s_i^2$, e che questa operazione richiede al più $O(\log_2^2(m))$ o.e. Una volta trovati tutti gli s_i occorre poi moltiplicarli tra di loro (tenendo conto degli esponenti $n_i = 0, 1$). Nel caso peggiore (ovvero quando tutti gli esponenti n_i sono uguali ad 1), vanno eseguiti esattamente k prodotti. Ognuno di questi prodotti in termini di o.e. costa circa $O(\log_2^2(m))$; si ha pertanto che il costo complessivo dell'elevamento alla potenza n è un

$$O\left(k \log_2^2(m)\right) = O\left(\log_2(n) \log_2^2(m)\right) = O\left(\text{size}\,(n)\,\text{size}\,(m)^2\right).$$

Osservazione 1. Poiché gli elementi di $\mathbb{Z}_m$ hanno ordine al più $\lambda(m)$ (o se si vuole $\varphi(m)$), se n è molto grande, si può pensare di rimpiazzare il valore di n con quello del suo residuo modulo $\lambda(m)$ (rispettivamente, modulo $\varphi(m)$). Pertanto, qualora si conoscesse il valore di $\lambda(m)$ (ad esempio perché è nota la fattorizzazione di m) si potrebbe rimpiazzare n con il suo residuo modulo $\lambda(m)$ (o $\varphi(n)$). In tal caso dovremmo eseguire un numero

minore di o.e. ed il costo di ogni potenza modulare di $\mathbb{Z}_m$ non supererebbe $O(\text{size}\,(m)^3)$. Molto spesso però i numeri che stanno alla base dei protocolli crittografici sono così elevati che determinare la loro fattorizzazione, come vedremo, è un problema tutt'altro che facile.

Esempio 1. Calcoliamo 3^{218} (mod 1000) utilizzando il Fast Powering Algorithm e stimando il numero di o.e. richieste. Per prima cosa rappresentiamo 218 in base 2:

$$218 = 2 + 2^3 + 2^4 + 2^6 + 2^7.$$

Allora 3^{218} diventa

$$3^{218} = 3^2 \cdot 3^{2^3} \cdot 3^{2^4} \cdot 3^{2^6} \cdot 3^{2^7}.$$

Adesso ci serve calcolare le potenze 3^{2^i} (mod 1000) fino ad $i = 7$. Queste sono esposte nella tabella 2.1, per creare la quale sono necessarie solamente 7 moltiplicazioni.

Tabella 2.1: Quadrati successivi di 3 modulo 1000.

i	0	1	2	3	4	5	6	7
3^{2^i} (mod 1000)	3	9	81	561	721	841	281	961

Da ultimo non ci resta che eseguire le restanti 4 moltiplicazioni

$$\begin{aligned} 3^{218} &= 3^2 \cdot 3^{2^3} \cdot 3^{2^4} \cdot 3^{2^6} \cdot 3^{2^7} \\ &\equiv 9 \cdot 561 \cdot 721 \cdot 281 \cdot 961 \pmod{1000} \\ &\equiv 489 \pmod{1000}. \end{aligned}$$

Si noti che, riducendo ogni volta i numeri modulo 1000 dopo ogni prodotto, non abbiamo mai dovuto operare con numeri estremamente grandi. In totale si sono svolte solamente 11 moltiplicazioni per calcolare 3^{218}, risparmiando parecchia energia rispetto al metodo naive.

2.6 Estrazioni di radici quadrate in campi finiti.

Siano dati un intero z ed un primo dispari p. Usando la Legge di reciprocità quadratica (Teoremi 1.18, 1.20 e l'algoritmo che ne deriva) si può determinare velocemente se z è un residuo quadratico modulo p, ovvero se l'equazione $x^2 = z + p\mathbb{Z}$ ammette soluzioni in $\mathbb{Z}_p$. Tuttavia questo modo di procedere non dice nulla su come trovare esplicitamente un'eventuale radice quadrata. Di seguito mostreremo una procedura che consente di determinare, in modo agevole, una radice quadrata per ogni residuo quadratico mod p.

Sia p un qualunque primo dispari e sia a un elemento non nullo di $\mathbb{Z}_p$ che sappiamo essere un quadrato. Vogliamo determinare un $r \in \mathbb{Z}_p$ tale che $r^2 = a$.

Per prima cosa ci serve avere un non-quadrato $y \in \mathbb{Z}_p$. Sebbene non esista in generale un algoritmo deterministico per trovare un non-quadrato in un campo finito, sappiamo che esattamente metà degli elementi di $\mathbb{Z}_p^*$ lo sono, pertanto dopo un numero sufficientemente grande di estrazioni non è difficile trovare un siffatto y (si veda anche l'osservazione 3). Scriviamo $p-1 = 2^k m$, con m dispari e calcoliamo $b := y^m$. È facile vedere che b è una radice primitiva 2^k-esima dell'unità, infatti poiché $b^{2^k} = y^{2^k m} = y^{p-1} = 1$, l'ordine di b divide 2^k, se poi fosse strettamente minore, diciamo 2^h con $h < k$, si avrebbe che

$$1 = (b^{2^h})^{2^{k-h-1}} = b^{2^{k-1}} = y^{2^{k-1}m} = y^{\frac{p-1}{2}},$$

e quindi, per la Proposizione 1.14, y dovrebbe essere un quadrato in $\mathbb{Z}_p$, in contraddizione con la nostra scelta. Dunque $|b| = 2^k$ e b è un generatore del 2-sottogruppo di Sylow, $\langle b \rangle$, del gruppo ciclico $\mathbb{Z}_p^*$.
A questo punto sia $r := a^{\frac{m+1}{2}}$ e consideriamo l'elemento l'elemento $r^2 a^{-1} = a^m$. Il suo ordine è un divisore di 2^{k-1}. Infatti, $(r^2 a^{-1})^{2^{k-1}} = a^{m 2^{k-1}} = a^{\frac{p-1}{2}} = 1$, per la Proposizione 1.14 essendo a un quadrato. In particolare abbiamo che $\langle r^2 a^{-1} \rangle$ è un 2-sottogruppo di $\mathbb{Z}_p^*$ e come tale è contenuto nel 2-Sylow $\langle b \rangle$ di $\mathbb{Z}_p^*$. Scritto $r^2 a^{-1} = b^s$, con $s \in (-2^k, 0]$, affermiamo che esiste un $j \in [0, 2^k)$ per cui risulta

$$a = (r b^j)^2. \tag{2.9}$$

Infatti occorre e basta osservare che s è pari. Sia γ una radice di a che stiamo cercando. Per come è definito m, l'elemento γ^m ha ordine una potenza di 2 e quindi $\gamma^m \in \langle b \rangle$, sia $\gamma^m = b^t$. D'altra parte abbiamo che

$$b^s = r^2 a^{-1} = a^m = \gamma^{2m} = (\gamma^m)^2 = b^{2t},$$

ovvero l'ordine di b, cioè 2^k, divide $s - 2t$, il che prova che s è pari. Scritto s come $s = -2j$, con $j \in [0, 2^{k-1})$ abbiamo che

$$r^2 a^{-1} = b^{-2j}$$

da cui segue subito la (2.9). Si tratta ora di trovare la giusta potenza $j \in [0, 2^{k-1})$ di b^j per cui $r b^j$ sia una radice quadrata di a. A tal fine scriviamo j in base 2 come

$$j = \sum_{i=0}^{k-2} j_i 2^i$$

(essendo $s = -2j > -2^k$), e tramite un procedimento induttivo mostriamo come determinare le cifre binarie di j.
<u>Passo 1.</u> Poiché $(r^2 a^{-1})^{2^{k-1}} = a^{\frac{p-1}{2}} = 1$ si ha che

$$(r^2 a^{-1})^{2^{k-2}} = \pm 1.$$

Se fosse $(r^2a^{-1})^{2^{k-2}} = 1$, prendiamo $j_0 = 0$, altrimenti sia $j_0 = 1$. Si noti che con questa scelta di j_0 si ha che $(b^{j_0}r)^2a^{-1}$ è una radice 2^{k-2}-esima dell'unità.

$$((b^{j_0}r)^2a^{-1})^{2^{k-2}} = b^{j_0 2^{k-1}} \cdot (r^2a^{-1})^{2^{k-2}} = \begin{cases} 1 \cdot 1 & \text{se } j_0 = 0 \\ -1 \cdot -1 & \text{se } j_0 = 1 \end{cases}$$

poiché $b^{2^{k-1}} = -1$ essendo $|b| = 2^k$.
<u>Passo 2.</u> Supponiamo d'aver trovato $j_0, j_1, \ldots, j_{h-1}$ tali che $(b^{\sum_{i=0}^{h-1} j_i 2^i} r)^2 a^{-1}$ è una radice 2^{k-h-1}-esima dell'unità, mostriamo come trovare j_h. Definiamo

$$j_h := \begin{cases} 0 \\ 1 \end{cases} \qquad \text{rispettivamente se} \quad (b^{\sum_{i=0}^{h-1} j_i 2^i} r)^2 a^{-1})^{2^{k-h-2}} = \begin{cases} 1 \\ -1 \end{cases}$$

Facilmente si verifica che, con questa scelta per j_h, si ha

$$(b^{\sum_{i=0}^{h} j_i 2^i} r)^2 a^{-1})^{2^{k-h-2}} = 1.$$

Quando h raggiunge $k-2$ si hanno tutti i coefficienti che definiscono j e otteniamo che

$$(b^j r)^2 a^{-1} = 1,$$

ovvero che $b^j r$ è una radice quadrata di a, come volevasi.

Stimiamo ora il costo computazione di questo algoritmo.
Supponiamo d'aver iniziato la procedura con già a disposizione un non-residuo quadratico y. I passaggi per trovare m, b ed $r = a^{\frac{m+1}{2}}$ (modulo p) richiedono al più $O(\log_2^3 p)$ o.e. (si veda la sezione sul costo delle potenze modulari). Per trovare j la parte più dispendiosa del processo induttivo consiste nell'eseguire le potenze 2^{k-j_i-2}-esime, questo significa fare $k - j_i - 2$ quadrati modulo p di interi minori di p. Poiché $k - j_i - 2 < k$, ogni potenza di questo tipo costa $O(k \log_2^2 p)$. In tutto ci sono $k-1$ cifre binarie da determinare e quindi $k-1$ passaggi da eseguire; il costo totale è pertanto un

$$O(log_2^3 p + k^2 \log_2^2 p) = O(\log_2^2 p(\log_2 p + k^2)).$$

Nel caso peggiore (se $p-1$ è una potenza di 2, ovvero p è un primo di Fermat[3]), si ha che il tutto è un $O(\log_2^4 p)$, essendo $k \approx \log_2 p$. In conclusione, dato un non-residuo modulo p, in tempo polinomiale è possibile fare l'operazione di estrazione di radice modulo p.

[3]Un primo p si dice di Fermat se $p = 2^a + 1$ per qualche $a \in \mathbb{N}$. Al momento sono noti solo cinque primi di Fermat, che sono

$$3,\ 5,\ 17,\ 257,\ 65537.$$

È una congettura aperta l'esistenza di una infinità di primi di Fermat.

Esempio 2. Calcoliamo le radici quadrate di $a = 562$ modulo $p = 601$.
Il primo non-residuo modulo 601 è $y = 7$. Abbiamo che $p - 1 = 600 = 2^3 \cdot 75$, quindi $b = 7^{75} = 59$ e $r = a^{38} = (562)^{38} = 74$ (dove per brevità rappresentiamo gli elementi di $\mathbb{Z}_{601}$ come gli interi). Dopo aver calcolato tramite l'algoritmo di Euclide $a^{-1} = 339$, notiamo che $a^{75} = r^2 a^{-1} = 74^2 \cdot 339 = 67 \cdot 339 = 476$ che è una radice quarta dell'unità. Calcoliamo ora $(r^2 a^{-1})^2 = (476)^2 = -1$, pertanto $j_0 = 1$. Ora calcoliamo $(br)^2 a^{-1} = (59 \cdot 74)^2 \cdot 339 = 39 \cdot 339 = -1$, ne segue che $j_1 = 1$, e pertanto $j = j_0 + j_1 \cdot 2 = 3$. Allora $b^3 r = (59)^3 \cdot 74 = 559$ è una radice di $a = 562$. L'altra radice è ovviamente $-559 = 42$.

Osservazione 2. Il caso più semplice dell'algoritmo sopra descritto è quando $p \equiv 3 \pmod 4$. Allora, con le notazioni di sopra, abbiamo che $k = 1$, $m = \frac{p-1}{2}$ e quindi $\frac{m+1}{2} = \frac{p+1}{4}$. Ne segue che $r = a^{\frac{p+1}{4}}$ è già una radice quadrata richiesta.

Osservazione 3. Come si diceva nel corso della presentazione dell'algoritmo, al momento non è noto (a meno di assumere l'Ipotesi di Riemann) nessun procedimento deterministico per trovare un non-residuo quadratico modulo p. Tuttavia, dato un qualunque $\epsilon > 0$ esiste un algoritmo polinomiale che trova un non-residuo con probabilità $> 1 - \epsilon$. Infatti scegliendo arbitrariamente y, con $0 < y < p$, si ha esattamente il 50% di possibilità che y sia un non-residuo mod p (e questo fatto può essere testato in tempo polinomiale calcolando il simbolo di Legendre per y modulo p). Se ripetiamo le scelte arbitrarie esattamente un numero di volte $\geq \log_2(1/\epsilon)$, allora con probabilità $> 1 - \epsilon$ riusciamo a trovare almeno un non quadrato $y + p\mathbb{Z} \in \mathbb{Z}_p$.

Osservazione 4. L'algoritmo che abbiamo presentato vale se sostituiamo $\mathbb{Z}_p$ con un qualunque campo finito $\mathbb{F}$. L'unica accortezza che dobbiamo porre riguarda il calcolo dei costi computazionali. Infatti a priori non sappiamo come venga implementata la moltiplicazione tra due elementi di $\mathbb{F}$. Possiamo però dire che detto ω il costo della moltiplicazione in $\mathbb{F}$, allora l'intero algoritmo impiega un $O(\omega \cdot \log_2^2 |\mathbb{F}|)$ o.e.

2.7 Equivalenza computazionale

Un *oracolo* per un problema computazionale P è un macchina ideale in grado di dare una risposta in tempo unitario ad ogni istanza del problema P. Si tratta semplicemente di una scatola nera che fornisce sempre e subito una soluzione al problema P, sia esso di tipo decisionale o funzionale.

Definizione 11. *Siano P_1 e P_2 due problemi computazionali e sia Θ un oracolo per P_2. Diremo che P_1* è riconducibile a P_2 in tempo polinomiale *se esiste un algoritmo per P_1 che usa Θ in modo tale che dette rispettivamente $g(n)$ ed $f(n)$ il numero di domande poste a Θ ed il numero di o.e. effettuate dall'algoritmo, si ha che esiste un $m \geq 0$ tale che*

$$f(n) + g(n) = O(\log_2^m n).$$

Se P_1 e P_2 sono riconducibili l'uno all'altro, allora P_1 e P_2 sono computazionalmente equivalenti.

L'idea è di voler descrivere la situazione in cui saper risolvere un problema permetta di risolverne anche un altro in tempo polinomiale. Facciamo subito un esempio, che tornerà anche utile nei capitoli successivi.

Sia n un numero intero prodotto di due primi distinti, p e q, ignoti e siano rispettivamente P_1 il problema di trovare una fattorizzazione per n e P_2 quello di trovare il valore di $\varphi(n)$. Allora esiste una equivalenza computazionale tra P_1 e P_2. Abbiamo già visto infatti che una volta risolto P_1, ovvero una volta nota la fattorizzazione $n = pq$, trovare il valore di $\varphi(n)$ è facile e consiste sostanzialmente nell'eseguire prodotti e quozienti, tutte operazioni che richiedono un tempo polinomiale. Viceversa, supponiamo di sapere il valore di $\varphi(n)$. Allora posto $n = pq$ (con per ora p e q numeri primi incogniti) sappiamo che

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

Pertanto le incognite p e q soddisfano sono una soluzione del sistema

$$\begin{cases} xy = n \\ x + y = n - \varphi(n) + 1 \end{cases}$$

In particolare p e q possono facilmente essere determinati risolvendo l'equazione

$$x^2 - (n - \varphi(n) + 1)x + n = 0,$$

cioè detto $b = n - \varphi(n) + 1$, trovando $\frac{b \pm \sqrt{b^2 - 4n}}{2}$. Per quanto visto in questo capitolo, tutte le operazioni di sopra sono di tipo polinomiale, quindi anche il problema P_1 è riconducibile in tempo polinomiale a P_2.

Capitolo 3

Sistemi crittografici classici

In questo Capitolo presentiamo gli schemi crittografici classici, o cosiddetti simmetrici, dove due interlocutori, dopo essersi prima privatamente accordati sulla scelta di un'unica chiave segreta, possono iniziare uno scambio di messaggi sicuro.

Definizione 12. *Un* sistema crittografico *è una quintupla* $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ *che soddisfa alle seguenti proprietà:*

1. $\mathcal{P}$, $\mathcal{C}$ *e* $\mathcal{K}$ *sono tre insiemi, chiamati rispettivamente: lo spazio dei testi in chiaro, quello dei testi cifrati e lo spazio delle chiavi (in inglese: the plaintext space, the ciphertext space, the key space).*

2. $\mathcal{E} = \{E_k | k \in \mathcal{K}\}$ *è una famiglia di funzioni iniettive*

$$E_k : \mathcal{P} \longrightarrow \mathcal{C}$$

   dette funzioni di codifica *(o encryption functions).*

3. $\mathcal{D} = \{D_k | k \in \mathcal{K}\}$ *è una famiglia di funzioni*

$$D_k : \mathcal{C} \longrightarrow \mathcal{P}$$

   dette funzioni di decodifica *(decryption functions).*

4. *Per ogni* $k \in \mathcal{K}$*, si richiede che esista un* $h \in \mathcal{K}$ *tale che*

$$D_h(E_k(p)) = p \quad \forall p \in \mathcal{P}.$$

L'idea alla base è la seguente. Due personaggi, convenzionalmente Alice e Bob, vogliono scambiarsi delle informazioni segrete tramite un canale pubblico non sicuro. Alice

manda un messaggio confidenziale $x \in \mathcal{P}$ a Bob usando pertanto uno schema crittografico. Per prima cosa, Alice sceglie una chiave $k \in \mathcal{K}$, con questa poi calcola $z = E_k(x)$, che invia a Bob. Bob riceve z e ottiene x, facendo $D_h(z)$. È ovvio che un tale sistema funziona finché la chiave di decodifica h rimane nota solo Bob.

L'esempio più semplice di schema crittografico è quello che si ottiene mediante una semplice *permutazione (monoalfabetica)*. Ovvero, si prende una permutazione π dell'alfabeto con cui si scrive il plaintext e si sostituisce ogni lettera del messaggio con la rispettiva immagine mediante π. La chiave di codifica è pertanto π, mentre quella di decodifica è la sua inversa π^{-1}. Forse il più antico cifrario a sostituzione monoalfabetica risale a Giulio Cesare. Secondo quanto afferma Svetonio nel *De vita Caesarum*, per la corrispondenza militare con le truppe di Quinto Tullio Cicerone, Giulio Cesare usava criptare i suoi messaggi mediante un semplice "shift di +3", ovvero sostituendo ogni lettera dell'alfabeto con la terza lettera seguente, quindi scrivendo D al posto di A, E al posto di B, e così via. Per quanto banale, all'epoca questo era un sistema abbastanza sicuro: gli avversari erano pressoché analfabeti e metodi di crittoanalisi in grado di rompere tale codice non esistevano. Volendo rappresentare un cifrario del tipo usato da Cesare con la notazione introdotta nella Definizione 12, dobbiamo innanzitutto identificare ogni lettera dell'alfabeto inglese Σ [1] con l'elemento di $\mathbb{Z}_{26}$ corrispondente, secondo la Tabella 3.1.

Tabella 3.1: Corrispondenza tra lettere dell'alfabeto inglese e numeri.

A	B	C	D	E	F	G	H	I	J	K	L	M
0	1	2	3	4	5	6	7	8	9	10	11	12
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
13	14	15	16	17	18	19	20	21	22	23	24	25

Fatto ciò, prendiamo $\mathcal{P} = \mathcal{C} = \mathcal{K} = \Sigma$, che, come detto, identifichiamo con $\mathbb{Z}_{26}$. Si sceglie poi una chiave di codifica $k \in \mathbb{Z}_{26}$ (nel caso dell'originale cifrario di Cesare si sceglie $k = 3$). Le funzioni di codifica e decodifica associate saranno pertanto

$$E_k : \Sigma \longrightarrow \Sigma \qquad\qquad D_k : \Sigma \longrightarrow \Sigma$$
$$x \longmapsto x + k \qquad\qquad x \longmapsto x - k$$

Come si vede, per questo cifrario la chiave di codifica e quella di decodifica coincidono. Non sempre è così, per ogni sistema crittografico.
Pensiamo che si voglia inviare il seguente messaggio

[1] Per semplicità e coerenza con i testi contemporanei di crittografia, consideriamo come lingua base l'inglese moderno.

"Gallia est omnis divisa in partes tres,"

dopo averlo criptato con uno shift a destra di 7 posizioni. Per ragioni pratiche, nel ciphertext vengono ignorati gli spazi, le punteggiature, così come non si fanno distinzioni tra lettere maiuscole e minuscole. Inoltre, il testo cifrato viene suddiviso in blocchi della stessa lunghezza, ad esempio blocchi di cinque lettere (l'ultimo blocco viene riempito con la stessa lettera, ad esempio Z). Il messaggio criptato da inviare è pertanto

```
NHSSP HLZAV TUPZK PCPZH PUWHY ALZAY LZZZZ
```

La Crittoanalisi

Il termine *crittoanalisi* deriva dalle parole greche *kryptós*, "nascosto" ed *analýein*, "scomporre". Con essa si intende infatti lo studio dei metodi che permettono di carpire il significato di informazioni cifrate, senza avere accesso direttamente alla chiave, in breve i modi per "attaccare un testo cifrato". Tipicamente la crittoanalisi rappresenta la controparte della crittografia. Nel corso della storia, ognuna ha contribuito (e contribuisce) direttamente al progresso dell'altra, creando un processo tipico dei fenomeni evolutivi detti della "Regina Rossa" che possiamo così enunciare:

> *"Un continuo miglioramento dell'adattamento è necessario per le specie anche solo per mantenere l'adattamento relativo all'ambiente in cui esso si è evoluto ed evolve."*[2]

Le cifrature mediante permutazione monoalfabetica sono in assoluto le più semplici da attaccare. Un modo "brutale", ma efficace, che consente di ricavare il testo originale dall'intercettazione di un testo cifrato col metodo di Cesare, senza conoscere la chiave, consiste nel decrittare il messaggio con ciascuna delle 26 chiavi possibili, finché non si trova qualcosa di sensato. Applicando questo sistema all'esempio precedente otteniamo

```
NHSSP HLZAV TUPZK PCPZH PUWHY ALZAY LZZZZ
OITTQ IMABW UVQAL QDQAI QVXIZ BMABZ MAAAA
PJUUR JNBCX VWRBM RERBJ RWYJA CNBCA NBBBB
...
GALLI AESTO MNISD IVISA INPAR TESTR ESSSS
```

Vista la semplicità del cifrario di Cesare, tale metodo è senz'altro efficace. La cosa invece non è affatto fattibile se come procedura di crittaggio viene utilizzata una qualunque permutazione monoalfabetica (che non sia necessariamente uno "shift"). Banalmente,

[2] La terminologia deriva da un passo del celebre racconto di Lewis Carroll, *Attraverso lo specchio e quel che Alice vi trovò.* "Ora, in questo luogo, come puoi vedere, ci vuole tutta la velocità di cui si dispone se si vuole rimanere nello stesso posto."

il numero di permutazioni è $26! \simeq 4,03 \cdot 10^{26}$, e ciò esclude a priori ogni metodo crittoanalitico basato su una ricerca esaustiva della chiave.
Furono gli arabi nel IX secolo a trovare un modo infallibile per attaccare questi cifrari classici. Tale procedura sfrutta la cosiddetta *analisi delle frequenze*. In ogni lingua la frequenza di uso di ogni lettera è piuttosto determinata. Nei testi lunghi questo è vero in modo rigoroso, ma spesso anche testi brevi hanno frequenze non molto diverse da quelle previste. Ad esempio, sia nell'italiano che nell'inglese la lettere E è quella comunemente più usata; in un tipico scritto italiano o inglese, sufficientemente grande, circa il 12% delle lettere che compaiono sono E. La Tabella 3.2 mostra le frequenze delle lettere in alcune lingue. Per attaccare uno schema crittografico basato sulla permutazione (monoalfabetica), per prima cosa si analizzano le frequenze delle lettere del testo cifrato e le si confrontano con i valori della Tabella 3.2. In questo modo si riesce a capire quale lettera è stata sostituita a quella dell'alfabeto di tale lingua. Se ad esempio in un testo la lettera più frequente è la X, allora ci saranno buone probabilità che tale lettera deve essere sostituita con la lettera E. Operando la sostituzione di tutte le lettere del testo cifrato secondo questo schema, il testo sarà in buona parte decriptato.

Similmente esistono anche tabelle di frequenza per digrammi e trigrammi (cioè blocchi di due e di tre lettere) che risultano essere molto utili per questo tipo di attacchi. I digrammi più usati in italiano sono nell'ordine: ER, ES, ON, RE, EL, EN, DE, mentre per l'inglese abbiamo: TH, HE, IN, ER, AN, RE. I trigrammi più ricorrenti in inglese sono: THE, ING, THA, AND, ION.

Come è logico aspettarsi, il mittente puó benissimo cercare delle difese dagli attacchi di questo tipo, ad esempio decidendo volontariamente di scrivere i propri messaggi senza usare una determinata lettera, o digramma/trigramma. Esistono esempi celebri di componimenti letterari (addirittura libri di centinaia di pagine) dove non compare mai una determinata lettera. Un lipogramma di trecento pagine molto famoso è *La Disparition* di Georges Perec, completamente scritto senza l'utilizzo della vocale "e", a cui ha fatto seguito *Les revenentes* nel quale invece Perec utilizza come sola vocale la "e" in tutto il testo. A noi piace anche ricordare il seguente tautogramma (componimento nel quale tutte le parole hanno la medesima lettera iniziale) e lipogramma (mancante della vocale "i") di Ilion (Nicola Aurelio) che nel 2003 vinse un concorso al 61° Congresso Nazionale di Enigmistica Classica.

FRAMMENTO FRANCESCANO

Fu fratello Francesco fresca fonte,
fuoco ferace fu: fede fruttuosa
fecelo forte. Fra fedele folla
fabulando fermò feroce fera.

Frale farfalla fu, ferrea fortezza
fra fosche fratte, fervente facella.
Fulcro fatato fu fenomenale,
fausta fortuna fra fatale falce.
Faro fornente futuro fastoso,
facondo formulò fato fulgente.

Flora, fauna, fondeva franco frate,
fuor facendo fantasma furfantesco,
fermando funeste forche.

Fu fante
fugante feral febbre flagellante,
fu fragranza flautata, falegname,
farmaco favoloso, focolare
fra fame fremebonda, fredde fosse.
Fasto fasullo falso frodatore,
frenò fuggendo forme forsennate,
fra fronzute foreste, forre fonde.

Fu frugale, festevole fu fabbro,
fu fanale fra feudale furore,
folletto folgorante:
fu Francesco.

Tabella 3.2: Frequenze delle lettere per alcune lingue europee (fonte [27]).

Lettera	in $\mathbb{Z}_{26}$	Inglese	Italiano	Spagnolo	Francese	Tedesco
A	0	8.17%	11.74%	12.53%	7.64%	6.51%
B	1	1.49%	0.92%	1.42%	0.90%	1.89%
C	2	2.78%	4.50%	4.68%	3.26%	3.06%
D	3	4.25%	3.73%	5.86%	3.67%	5.08%
E	4	12.70%	11.79%	13.68%	14.71%	17.40%
F	5	2.23%	0.95%	0.69%	1.07%	1.66%
G	6	2.01%	1.64%	1.01%	0.87%	3.01%
H	7	6.09%	1.54%	0.70%	0.73%	4.76%
I	8	6.97%	11.28%	6.25%	7.53%	7.55%
J	9	0.15%	-	0.44%	0.55%	0.27%
K	10	0.77%	-	0.01%	0.05%	1.21%
L	11	4.02%	6.51%	4.97%	5.46%	3.44%
M	12	2.41%	2.51%	3.15%	2.97%	2.53%
N	13	6.75%	6.88%	6.71%	7.10%	9.78%
O	14	7.51%	9.83%	8.68%	5.38%	2.51%
P	15	1.93%	3.05%	2.51%	3.02%	0.79%
Q	16	0.09%	0.51%	0.88%	1.36%	0.02%
R	17	5.99%	6.37%	6.87%	6.55%	7.00%
S	18	6.33%	4.98%	7.98%	7.95%	7.27%
T	19	9.06%	5.62%	4.63%	7.24%	6.15%
U	20	2.76%	3.01%	3.93%	6.31%	4.35%
V	21	0.98%	2.10%	0.90%	1.63%	0.67%
W	22	2.36%	-	0.02%	0.11%	1.89%
X	23	0.15%	-	0.22%	0.39%	0.03%
Y	24	1.97%	-	0.90%	0.31%	0.04%
Z	25	0.07%	0.49%	0.52%	0.14%	1.13%
Ñ		-	-	0.31%	-	-
À		-	-	-	0.49%	-
Ç		-	-	-	0.09%	-
È		-	-	-	0.27%	-
É		-	-	-	1.90%	-
Ê		-	-	-	0.23%	-
Ë		-	-	-	0.00%	-
Î		-	-	-	0.05%	-
Ï		-	-	-	0.01%	-
Ù		-	-	-	0.06%	-
Œ		-	-	-	0.02%	-
ß		-	-	-	-	0.31%

3.1 Cifrature a blocco

Definizione 13. *Un crittosistema sull'alfabeto Σ è chiamato* cifratura a blocco *se plaintext space e cyphertext space coincidono con Σ^k, l'insieme di tutte le parole di lunghezza fissata k, per qualche $k \in \mathbb{N}_{>0}$ detto* ampiezza del blocco.

L'esempio più semplice è ovviamente il già citato cifrario di Cesare, dove $k = 1$.

3.1.1 Il cifrario di Vigenère

Nel 1586 un diplomatico, traduttore ed alchimista francese di nome Blaise de Vigenère rese pubblico il primo esempio di cifrario polialfabetico. In realtà l'idea originaria di questo cifrario fu introdotta per la prima volta nel 1553 da Giovan Battista Bellaso, nell'opera *La cifra del Sig. Giovan Battista Belaso*, ed è piuttosto semplice: invece di spostare sempre dello stesso numero di posti la lettera da cifrare, come accade nel cifrario di Cesare, questa viene spostata di un numero di posti variabile ma ripetuto, determinato in base ad una parola chiave, da concordarsi tra mittente e destinatario. In concreto, detto $\Sigma = \mathbb{Z}_n$ l'alfabeto, Alice e Bob concordano privatamente una chiave segreta, ovvero un numero naturale $k \in \mathbb{N}_{>0}$ ed un vettore $\underline{v} = (v_1, v_2, \ldots, v_k) \in \Sigma^k$. Ora, se Alice vuole inviare un messaggio criptato a Bob per prima cosa rende tale messaggio una stringa di lunghezza multipla di k, diciamo mk per qualche $m \in \mathbb{N}$ (ad esempio completando il messaggio con simboli privi di significato). Pertanto, si assume che $\mathcal{P} = \mathcal{C} = \Sigma^{km}$. Quindi Alice usa la seguente funzione di codifica

$$\begin{aligned} E_{\underline{v}} : \mathcal{P} &\longrightarrow \mathcal{C} \\ \underline{x} &\longmapsto \underline{x} + \tilde{\underline{v}}, \end{aligned}$$

dove $\tilde{\underline{v}}$ è il vettore di Σ^{mk} che si ottiene tramite giustapposizione di m copie del vettore $\underline{v}$ (ovvero l'immagine della componente x_j di vettore $\underline{x}$ è $x_j + v_{i_j}$, dove i_j è il residuo di j modulo k). La funzione di decodifica per Bob è ovviamente

$$\begin{aligned} D_{\underline{v}} : \mathcal{C} &\longrightarrow \mathcal{P} \\ \underline{z} &\longmapsto \underline{z} - \tilde{\underline{v}}. \end{aligned}$$

Crittoanalisi di Vigenère

Per quasi tre secoli questo cifrario ha goduto di enorme successo, al punto da essere chiamato, a torto, con l'appellativo di *"le chiffre indéchiffrable"*. Un primo efficace metodo di decrittazione comparve nel 1863 per opera di un colonnello prussiano Friedrich

Kasiski. È certo però che Vigenère fu già forzato intorno al 1840 da Charles Babbage, che non pubblicò mai i risultati della sua ricerca[3].

La debolezza del cifrario di Vigenère sta nell'essere, di fatto, un insieme di k cifrari di Cesare, dove k è la lunghezza della chiave. Il primo passo che un crittoanalista deve fare è quindi determinare la lunghezza della chiave. Di seguito presentiamo oltre al test di Zariski, il più moderno metodo di crittoanalisi dovuto a William F. Friedman, nel quale si fa uso di importanti nozioni di calcolo delle probabilità.

Il test di Zasiski

Questo metodo di crittoanalisi, come il seguente, si divide in due fasi, nella prima si determina la lunghezza k della chiave, nella seconda la chiave segreta $\underline{v}$. Per determinare k, il test di Zariski suggerisce di cercare nel cyphertext $\underline{z}$ le sequenze di digrammi, o meglio di trigrammi, che si ripetono più spesso. Dopodiché per ogni siffatto trigramma si va a misurare le distanze che intercorrono tra le varie ripetizioni, contando i caratteri che separano rispettivamente la prima lettera del trigramma dalle prime lettere dei suoi ripetuti. A questo punto si calcola d, il massimo comune divisore fra tutte queste distanze e si congettura che k sia un divisore di d. Se il messaggio di testo intercettato è abbastanza lungo (rispetto alla lunghezza della chiave) ci sono buone probabilità di trovare in questo modo k. Una volta trovato k, inizia la seconda fase della crittoanalisi. Il cyphertext z viene diviso nei k blocchi

$$\underline{z}^i = z_i z_{i+k} z_{i+2k} \dots \qquad \text{per ogni } i = 1, 2, \dots, k$$

e a ciascuno di questi viene applicata l'analisi delle frequenze per determinare la componente v_i della chiave.

Il test di Friedman

Definizione 14. *Siano $\underline{x} = x_1 \dots x_m$ ed $\underline{y} = y_1 \dots y_l$ due stringhe di testo nell'alfabeto $\Sigma = \mathbb{Z}_n$. Si definisce* indice di mutua coincidenza *la probabilità che presi arbitrariamente un carattere da $\underline{x}$ ed uno da $\underline{y}$ questi coincidano. Si indica con $IM_c(\underline{x}, \underline{y})$. Se $\underline{x} = \underline{y}$, allora si scrive $I_c(\underline{x})$ in luogo di $IM_c(\underline{x}, \underline{x})$ e si parla di indice di coincidenza di $\underline{x}$.*

L'importanza di questo concetto risiede nel fatto che attraverso l'indice di coincidenza è possibile capire se una data stringa di testo sia casuale oppure sia la cifrata, tramite una permutazione (monoalfabetica), di una stringa di testo di senso compiuto. Vediamo come.

Incominciamo con l'osservare che dalla definizione segue automaticamente che se σ

[3] Charles Babbage (1791–1871) è stato un matematico e filosofo britannico. Per primo ideò e progettò un calcolatore programmabile, da lui chiamato *"macchina delle differenze"*.

è una qualunque permutazione dell'alfabeto Σ, allora $I_c(\underline{x}) = I_c(\sigma(\underline{x}))$ (similmente $IM_c(\underline{x}, \underline{y}) = IM_c(\sigma(\underline{x}), \sigma(\underline{y})))$, questo semplicemente poiché le definizione di $I_c(\underline{x})$ (e $IM_c(\underline{x}, \underline{y})$) tengono conto di tutte le lettere dell'alfabeto.
Ora per ogni lettera $i \in \mathbb{Z}_n$, indichiamo con $f_i(\underline{x})$ il numero di volte che il simbolo i compare nella stringa $\underline{x}$, ovvero

$$f_i(\underline{x}) = |\{j | x_j = i\}| \,.$$

L'indice di coincidenza di $\underline{x} = x_1 x_2 \ldots x_m$ è pertanto dato dalla formula

$$I_c(\underline{x}) = \sum_{i=0}^{n-1} \frac{\binom{f_i(\underline{x})}{2}}{\binom{m}{2}} = \binom{m}{2}^{-1} \cdot \sum_{i=0}^{n-1} \binom{f_i(\underline{x})}{2},$$

dove m è la lunghezza della stringa $\underline{x}$. Adesso indichiamo con P_i la frequenza della lettera i nel linguaggio che stiamo considerando (i valori P_i per le principali lingue europee sono espressi nella tabella 3.2). La probabilità che in un testo di senso compiuto si abbiano almeno due occorrenze della lettera i è pertanto prossima a P_i^2. Possiamo quindi definire, per ogni linguaggio, un *indice di coincidenza standard*, dato da

$$I_c := \sum_{i=0}^{n-1} P_i^2.$$

(Nel caso dell'inglese moderno tale numero è circa $0,065$, mentre per l'italiano si approssima con $0,075$). Ne segue che se $\underline{z}$ è una stringa di senso compiuto sufficientemente grande, allora il valore di $I_c(\underline{z})$ non deve discostarsi molto dal valore standard I_c. Ma vale molto di più! Infatti se la nostra stringa di base $\underline{x}$, che vogliamo decifrare, proviene da un testo $\underline{z}$ di senso compiuto tramite una qualche permutazione σ di Σ, allora per quanto detto

$$I_c(\underline{x}) = I_c(\sigma(\underline{z})) \approx I_c,$$

cioè

$$\binom{m}{2}^{-1} \cdot \sum_{i=0}^{m} \binom{f_i(\underline{x})}{2} \approx \sum_{i=0}^{n-1} P_i^2.$$

Nel caso invece in cui $\underline{x}$ non è la cifrata di un testo effettivo, allora tutti i simboli sono equiprobabili (con probabilità $1/n$) per le cifre di $\underline{x}$ e quindi $I_c(\underline{x})$ sarà prossimo a

$$\sum_{i=0}^{n-1} \frac{1}{n^2} = \frac{1}{n} \approx 0,038.$$

In questo modo si può pertanto capire se $\underline{x}$ è una stringa casuale oppure è la cifrata di qualche testo di senso compiuto.

Supponiamo ora che Alice invii a Bob il messaggio $\underline{z} = z_1 z_2 \dots z_m$, che è stato cifrato col sistema di Vigenére scegliendo come chiave segreta $\underline{v} = v_1 v_2 \dots v_k$. Eva intercetta $\underline{z}$ e vuole per prima cosa determinare k, la lunghezza della chiave. Allora fissa un intero h (in genere decide di partire da $h = 2$ per poi salire) e considera le h stringhe

$$\underline{z}^{(i)} = z_i z_{i+h} z_{i+2h} \dots \qquad \text{per ogni } i = 1, 2, \dots, h.$$

Per queste calcola i valori $I_c(\underline{z}^{(1)}), I_c(\underline{z}^{(2)}), \dots I_c(\underline{z}^{(h-1)})$. Se questi valori si discostano molto da I_c, allora la scelta di h come possibile lunghezza della chiave era sbagliata, pertanto sceglie un nuovo intero h e ripete l'operazione. Quando si ha che tutti i valori di $I_c(\underline{z}^{(1)}), I_c(\underline{z}^{(2)}), \dots I_c(\underline{z}^{(h-1)})$ sono prossimi all'indice di coincidenza standard, allora ci sono ottime probabilità che sia $h = k$. Per facilitare il lavoro e non procedere alla cieca, Eva può prima congetturare quale possa essere la lunghezza della chiave tramite il test di Zariski e poi verificare tale supposizione calcolando gli indici di coincidenza.

Una volta trovata la lunghezza k, Eva si preoccupa di determinare la chiave $\underline{v}$ e per fare questo utilizza l'indice di mutua coincidenza. Come prima prende in esame le k stringhe

$$\underline{z}^{(i)} = z_i z_{i+k} z_{i+2k} \dots \qquad \text{per ogni } i = 1, 2, \dots, k.$$

Ora per ogni lettera j dell'alfabeto $\Sigma = \mathbb{Z}_n$ si ha che ogni volta che j compare come carattere nella stringa $\underline{z}^{(i)}$, questo è il risultato di uno shift di $+v_i$. Pertanto se chiamiamo $P_j^{(i)}$ la frequenza probabilistica con cui j compare in $\underline{z}^{(i)}$, essendo $\underline{z}^{(i)}$ la codifica di una stringa di senso compiuto, si ha che

$$P_j^{(i)} \approx P_{j-v_i}.$$

Con ovvie notazioni abbiamo che

$$IM_c(\underline{z}^{(1)}, \underline{z}^{(2)}) = \sum_{j=0}^{n-1} P_j^{(1)} P_j^{(2)} \approx \sum_{j=0}^{n-1} P_{j-v_1} P_{j-v_2} = \sum_{j=0}^{n-1} P_j P_{j+v_1-v_2}.$$

Più in generale, per ogni lettera $h_1 \in \mathbb{Z}_n$ fissata, Eva definisce $\underline{z}^{(1)}(h_1)$ come la stringa

$$\underline{z}^{(1)}(h_1) := x_1 + h_1, x_{1+k} + h_1, x_{1+2k} + h_1, \dots$$

e calcola

$$IM_c\big(\underline{z}^{(1)}(h_1), \underline{z}^{(2)}\big) \approx \sum_{j=0}^{n-1} P_{j-(v_1+h_1)} P_{j-v_2} = \sum_{j=0}^{n-1} P_j P_{j+v_1+h_1-v_2}. \tag{3.1}$$

Il trucco che Eva ora utilizza consiste nel calcolare $IM_c\big(\underline{z}^{(1)}(h_1), \underline{z}^{(1)}\big)$, al variare di h_1, fino a che non trova un valore di h_1 per cui tale numero sia assai prossimo all'indice di coincidenza I_c caratteristico della lingua. Questo infatti vorrebbe esattamente dire che

$$v_1 + h_1 - v_2 = 0$$

ovvero che

$$v_2 - v_1 = h_1.$$

Dopodiché ripete questo procedimento considerando $IM_c(\underline{z}^{(1)}(h_2), \underline{z}^{(3)})$ al variare di $h_2 = 0, 1, 2, \ldots, n-1$. Quando $IM_c\big(\underline{z}^{(1)}(h_2), \underline{z}^{(3)}\big) \approx I_c$ allora $v_3 - v_1 = h_2$. Poi passa a terminare h_3 e così via. Procedendo in questo modo alla fine Eva riesce ad ottenere un sistema di $k-1$ equazioni in k incognite

$$\begin{cases} v_2 - v_1 = h_1 \\ v_3 - v_1 = h_2 \\ \ldots \\ v_k - v_1 = h_{k-1} \end{cases}$$

Pone tutte le variabili in funzione di una sola incognita, diciamo v_1, e da ultimo prova tutti i possibili valori per v_1, che sono n, fino a quando il testo non diventa qualcosa di sensato.

3.1.2 Cifrature a blocchi affini lineari

Una cifratura a blocchi, con blocchi di lunghezza k e con plaintext-space $\mathcal{P}$ e ciphertext-space $\mathcal{C}$ uguali a $(\mathbb{Z}_n)^{mk}$, (con $m \geq 1$) si dice *affine lineare* se tutte le funzioni di codifica e decodifica sono affini lineari, ovvero se per ogni chiave $h \in \mathcal{K}$ si ha che

$$\begin{aligned} E_h : (\mathbb{Z}_n)^{mk} &\longrightarrow (\mathbb{Z}_n)^{mk} \\ \underline{x} &\longmapsto A\underline{x} + \underline{b} \end{aligned}$$

dove A e $\underline{b}$ sono elementi rispettivamente di $GL(mk, \mathbb{Z}_n)$ e $(\mathbb{Z}_n)^{mk}$. Ricordiamo che la notazione $GL(mk, \mathbb{Z}_n)$ indica il *gruppo generale lineare su* $\mathbb{Z}_n$, ovvero l'insieme delle matrici quadrate $mk \times mk$, ad entrate in $\mathbb{Z}_n$, che sono invertibili. Una matrice $A \in GL(mk, \mathbb{Z}_n)$ è invertibile se e solo se il suo determinante, $\det A$, è un elemento invertibile di $\mathbb{Z}_n$, ovvero se solo se $\det(A)$ è coprimo con n. La chiave del sistema crittografico è pertanto rappresentata dalla coppia $h := (A, \underline{b})$, e dunque $\mathcal{K} = GL(mk, \mathbb{Z}_n) \times (\mathbb{Z}_n)^{mk}$. La funzione di decodifica associata ad E_h è

$$\begin{aligned} D_h : (\mathbb{Z}_n)^{mk} &\longrightarrow (\mathbb{Z}_n)^{mk} \\ \underline{z} &\longmapsto A^{-1}(\underline{z} - \underline{b}) \end{aligned}$$

dove A^{-1} indica l'inversa della matrice A.

Esempi.

1. Quando $(A, b) = (Id, \tilde{\underline{v}})$ per un qualche $\underline{v} \in \mathbb{Z}_n^k$, si ottiene esattamente un cifrario di Vigenère.

2. Se invece $(A, \underline{b}) = (A, \underline{0})$, al variare di A in $GL(mk, \mathbb{Z}_n)$ si parla di *cifrario di Hill* (da Lester Hill che li introdusse nel 1929).

3. Ogni cifrario di permutazione (o anagramma) è un esempio di cifrario lineare. Sia data infatti una arbitraria permutazione π su mk oggetti e sia E_π la funzione di codifica definita ponendo per ogni $\underline{x} = (x_1, x_2, \ldots, x_{mk})$,
$$E_\pi(\underline{x}) = (x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(mk)}).$$
Allora la matrice (di permutazione) $M_\pi = (m_{ij})_{i,j}$, definita ponendo $m_{ij} = 1$ se $\pi(i) = j$, altrimenti $m_{ij} = 0$, è tale che
$$E_\pi(\underline{x}) = M_\pi \cdot \underline{x}.$$

Crittoanalisi delle cifrature a blocco affini lineari

Rompere un cifrario a blocchi affini lineari non è affatto un'operazione difficile. Mostreremo che è sufficiente conoscere una quantità relativamente piccola di plaintext per scoprire la chiave di cifratura, e quindi decrittare tutti i messaggi. Per questo motivo se vogliamo utilizzare un cifrario a blocchi sicuro, dobbiamo accertarci che questo non sia affine lineare, meglio che le sue funzioni di codifica/decodifica siano non facilmente approssimabili tramite funzioni affini lineari.

Sia al solito k la dimensione dei blocchi, scriviamo l in luogo di mk e siano
$$\begin{aligned} E_{(A,\underline{b})} : (\mathbb{Z}_n)^l &\longrightarrow (\mathbb{Z}_n)^l \\ \underline{x} &\longmapsto A\underline{x} + \underline{b} \end{aligned} \qquad \begin{aligned} D_{(A,\underline{b})} : (\mathbb{Z}_n)^l &\longrightarrow (\mathbb{Z}_n)^l \\ \underline{z} &\longmapsto A^{-1}(\underline{z} - \underline{b}) \end{aligned}$$
le funzioni di codifica e decodifica rispetto alla chiave $(A, \underline{b}) \in GL(l, \mathbb{Z}_n) \times (\mathbb{Z}_n)^l$ scelta da Alice e Bob. Supponiamo che Eva sia riuscita in qualche modo a decifrare $l + 1$ plaintext $\underline{x}_i$, per $i = 0, 1, \ldots, l$, che corrispondono rispettivamente ai messaggi crittati $\underline{z}_i = A\underline{x}_i + \underline{b}$, $i = 0, 1, \ldots, l$. Pertanto Eva conosce le l equazioni vettoriali
$$\underline{z}_i - \underline{z}_0 = A(\underline{x}_i - \underline{x}_0) \quad \text{per } i = 1, 2, \ldots, l. \tag{3.2}$$
Chiama X e Z le matrici quadrate $l \times l$
$$X = (\underline{x}_1 - \underline{x}_0 | \underline{x}_2 - \underline{x}_0 \ldots | \underline{x}_l - \underline{x}_0) \quad \text{e} \quad Z = (\underline{z}_1 - \underline{z}_0 | \underline{z}_2 - \underline{z}_0 \ldots | \underline{z}_l - \underline{z}_0),$$

le cui colonne i-esime sono rispettivamente i vettori $\underline{x}_i - \underline{x}_0$ e $\underline{z}_i - \underline{z}_0$. Il sistema (3.2) allora equivale a

$$Z = A \cdot X.$$

A questo punto Eva calcola $\det(X)$ e se questo è coprimo con n (cosa che accade con probabilità $\frac{\varphi(n)}{n}$, piuttosto alta) la matrice X è invertibile ed Eva trova la prima parte della chiave facendo

$$A = Z \cdot X^{-1}.$$

Una volta nota la matrice A, Eva risale a $\underline{b}$ tramite

$$\underline{b} = \underline{z}_0 - A\underline{x}_0.$$

In conclusione, la chiave del crittosistema è stata determinata dalla sola conoscenza di $l + 1$ plaintext-cyphertext. Nel caso dei cifrari di Hill ne bastano l, poiché si ha che $\underline{z}_0 = \underline{x}_0 = \underline{b} = \underline{0}$.

3.1.3 Modalità di funzionamento di alcuni cifrari a blocchi

Discutiamo ora brevemente l'utilizzo delle cifrature a blocco per la codifica di documenti arbitrariamente lunghi.

Col termine di modalità di funzionamento (in Inglese: "modes of operation") si indica un algoritmo che usa un cifrario a blocchi per garantire la sicurezza delle informazioni, quali soprattutto confidenzialità e autenticità. Un cifrario a blocchi di per sé è solo in grado di gestire la trasmissione sicura (codifica o decodifica) di un gruppo di bit di lunghezza fissata. Una modalità di funzionamento invece descrive come applicare ripetutamente la singola operazione del cifrario a blocchi per trasformare in modo sicuro una quantità di dati di gran lunga maggiore rispetto a quella del singolo blocco.
Le prime modalità di funzionamento dei cifrari a blocchi sono state definite dal NIST (National Institute of Standards and Technology) nel documento "FIPS 81" del 1981 [15]. Si tratta in tutto di quattro modes, i cui acronimi sono: ECB, CBC, OFB e CFB. In seguito il NIST ha ampliato la lista ad altre modalità. Noi vedremo in dettaglio solo le prime due modalità (il lettore interessato può consultare il libro [8], 3.8).

ECB, Electronic CodeBook Mode

Indichiamo con Σ un alfabeto e sia k un intero positivo. Considereremo blocchi di ampiezza k. Supponiamo quindi che $\mathcal{P} = \mathcal{C} = \Sigma^{mk}$ per qualche $m \geq 1$, e indichiamo con

$$\underline{x} = x_1 x_2 \ldots x_m$$

il generico messaggio in chiaro, dove $x_i \in \Sigma^k$ sono i singoli blocchi. Scelte le chiavi $h, d \in \mathcal{K}$ le funzioni di codifica e decodifica sono pertanto definite da

$$E_h : \Sigma^k \longrightarrow \Sigma^k \quad \text{e} \quad D_d : \Sigma^k \longrightarrow \Sigma^k$$

Il messaggio cifrato consiste semplicemente nella giustapposizione delle cifrature dei blocchi, ovvero

$$\underline{z} = E_h(x_1)E_h(x_2)\dots E_h(x_m).$$

Similmente accade per la decodifica.
Schematicamente la modalità ECB si rappresenta così

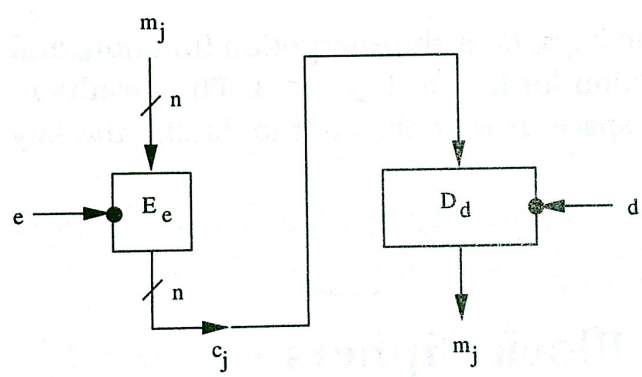


Figura 3.1: ECB

Esempio 3. Siano $\Sigma = \mathbb{Z}_2$, $k = 4$ e $\mathcal{K} = S_4$ il gruppo simmetrico su 4 oggetti. Presa $h = \pi$ definiamo

$$\begin{aligned} E_\pi : \{0,1\}^4 &\longrightarrow \{0,1\}^4 \\ x_1x_2x_3x_4 &\longmapsto x_{\pi(1)}x_{\pi(2)}x_{\pi(3)}x_{\pi(4)} \end{aligned}$$

e similmente per $D_{\pi^{-1}}$. Supponiamo di voler criptare il messaggio

$$101100010100101.$$

Per prima cosa rendiamo questo messaggio un elemento di $\mathcal{P}$ aggiungendo uno zero alla destra, ovvero definendo

$$\underline{x} = 1011.0001.0100.1010 \in \mathcal{P} = \Sigma^{mk}$$

(con $m = k = 4$). I blocchi di $\underline{x}$ sono pertanto

$$x_1 = 1011, \quad x_2 = 0001, \quad x_3 = 0100, \quad x_4 = 1010.$$

Scelta la chiave $\pi = (1234)$ e applicate le funzioni di codifica E_π, otteniamo

$$z_1 = E_\pi(x_1) = 0111, \quad z_2 = E_\pi(x_2) = 0010, \quad z_3 = E_\pi(x_3) = 1000, \quad z_4 = E_\pi(x_4) = 0101,$$

pertanto la cifratura consiste in

$$\underline{z} = 0111001010000101.$$

ECB presenta diversi svantaggi: uno fra tutti il fatto che blocchi uguali di plaintext vengono criptati nello stesso modo. Questo apre la strada ad attacchi di tipo statistico. Non è pertanto una modalità di cifratura consigliata per documenti lunghi.

CBC, CipherBlock Chaining Mode

A differenza della modalità precedente, in CBC la cifratura di un blocco dipende sia dalla chiave scelta che dal blocco precedente. Vediamo come funziona.

Per la codifica si sceglie un vettore di iniziazione, che chiamiamo IV, per "initialization vector"

$$IV \in \Sigma^k.$$

Detto quindi $\underline{x} = x_1 x_2 \dots x_m$ il messaggio da criptare, con $x_i \in \Sigma^k$, si pone

$$\begin{aligned} z_0 &= IV \\ z_j &= E_h(z_{j-1} \oplus x_j) \quad \text{per } j = 1, 2, \dots, m. \end{aligned}$$

(dove il simbolo $\oplus$ indica la somma "XOR" in Σ^k ed è definito da $a \oplus b = (a_1 + b_1, a_2 + b_2, \dots, a_k + b_k)$ se $a = (a_1, a_2, \dots, a_k)$ e $b = (b_1, b_2, \dots, b_k)$).

La decodifica avviene ponendo

$$\begin{aligned} z_0 &= IV \\ x_j &= z_{j-1} \oplus D_d(z_j) \quad \text{per } j = 1, 2, \dots, m. \end{aligned}$$

dove $D_d(E_h(w)) = w$ per ogni $w \in \Sigma^k$. Infatti abbiamo che

$$z_{j-1} \oplus D_d(z_j) = z_{j-1} \oplus D_d(E_h(z_{j-1} \oplus x_j)) = z_{j-1} \oplus (z_{j-1} \oplus x_j) = (z_{j-1} \oplus z_{j-1}) \oplus x_j = x_j$$

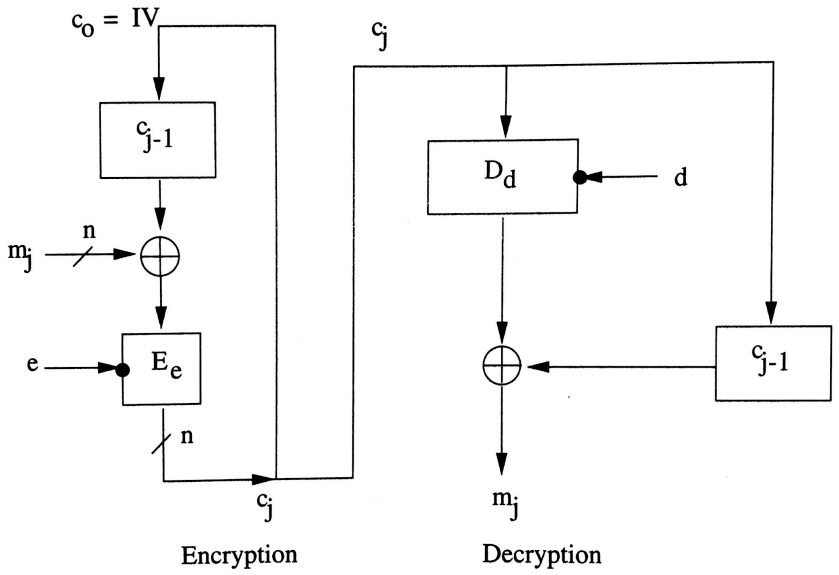


Figura 3.2: CCB

Esempio 4. Prendiamo gli stessi dati dell'esempio precedente, ovvero

$$x_1 = 1011, \quad x_2 = 0001, \quad x_3 = 0100, \quad x_4 = 1010$$

e $\pi = (1234)$ come chiave di cifratura. Sia inoltre $IV = 1010$.
Otteniamo $z_0 = IV = 1010$, $z_1 = E_\pi(z_0 \oplus x_1) = E_\pi(0001) = 0010$, $z_2 = E_\pi(z_1 \oplus x_2) = E_\pi(0011) = 0110$, ... e quindi in conclusione

$$\underline{z} = 0010011001001101.$$

Come si vede, in generale parti di plaintext uguali vengono criptate in modo diverso usando CBC. Poiché la codifica di un blocco dipende anche dal blocco precedente e non solo dalla chiave, se l'ordine dei blocchi di un messaggio cifrato viene cambiato, o se un blocco viene perso, la decodifica diventa impossibile.

Analizziamo ora come CBC si comporta per quanto riguarda la trasmissione degli errori. Durante il processo di decodifica il blocco di plaintext x_j viene computato grazie ai blocchi z_j e z_{j-1}. Pertanto se il blocco z_j viene trasmesso in modo incorretto, allora i blocchi x_j e x_{j+1} saranno decifrati incorrettamente. Ma la restante parte di plaintext $x_{j+2}, x_{j+3}, \ldots$ non sarà influenzata da questo errore e si potrà determinare in modo corretto.

Capitolo 4

Crittografia a chiave pubblica

La crittografia a chiave pubblica si fonda sul concetto di funzione *"one-way"*.

Definizione 15. *Una biezione $f : A \longrightarrow B$ si dice* one-way *se è facile computare f ed è estremamente difficile computare la sua inversa $f^{-1} : B \longrightarrow A$ (o almeno così si ritiene che sia). Inoltre f viene detta* trapdoor one-way *se f è one-way e nel contempo esiste un 'escamotage'* [1] *conoscendo il quale il computo di f^{-1} risulta facile.*

Proprio perché la natura computazionale di una funzione one-way è completamente discorde da quella della sua inversa, la crittografia a chiave pubblica viene anche spesso chiamata *crittografia asimmetrica*.

Vediamo formalmente come si presenta un crittosistema a chiave pubblica.
Indichiamo con $\mathcal{P}$ il plaintext space, $\mathcal{C}$ il ciphertext space e con $\mathcal{K}$ lo spazio delle chiavi. Siano inoltre

$$E : \mathcal{P} \times \mathcal{K} \longrightarrow \mathcal{C} \quad \text{e} \quad D : \mathcal{C} \times \mathcal{K} \longrightarrow \mathcal{P}$$

rispettivamente le funzioni di codifica e decodifica, che soddisfanno alla seguente relazione

(**CP1**) $D(E(x, k), k) = x$, per ogni $x \in \mathcal{P}$ e ogni $k \in \mathcal{K}$.

Questa dice semplicemente che l'operazione di codifica seguita dalla codifica con la stessa chiave produce il messaggio originale.
Ora, i primi requisiti della crittografia a chiave pubblica si possono enunciare con i seguenti assiomi.

(**CP2**) Calcolare E è una procedura facile.

[1] letteralmente trapdoor significa botola.

(**CP3**) Calcolare D è una procedura difficile.

(Dove, qui e nel seguito, col termine 'facile' si intende in tempo polinomiale, mentre 'difficile' sta per NP-completo. Nella pratica però il significato è sempre qualcosa di meno preciso.)

Vediamo il significato di ciò. Assumiamo che Alice invii il messaggio cifrato z a Bob e che Eva, l'intercettatrice, conosca non solo z, ma anche la chiave k e le funzioni E ed D usate per la codifica e la decodifica. Per decrittare z ad Eva basta eseguire l'operazione $D(z, k)$. Tuttavia, questo per lei è un problema difficile; si assume che anche, con la più sofisticata tecnologia, il tempo stimato per eseguire questa operazione è più lungo della vita dei protagonisti.
Sorge allora la domanda spontanea. Se la decodifica è difficile, come fa Bob a leggere i messaggi di Alice? La risposta della crittografia a chiave pubblica è la seguente. Esiste un insieme $\mathcal{S}$, detto di *chiavi segrete*, ed una coppia di funzioni l'una l'inversa dell'altra

$$g : \mathcal{S} \longrightarrow \mathcal{K} \quad \text{e} \quad h : \mathcal{K} \longrightarrow \mathcal{S}$$

tali che

(**CP4**) Calcolare $D^*(z, s) := D(z, g(s))$ è facile per ogni $z \in \mathcal{C}$ ed ogni $s \in \mathcal{S}$.

(**CP5**) Calcolare g è facile.

(**CP6**) Calcolare h è difficile.

L'assioma (**CP4**) afferma che, dati $s \in \mathcal{S}$ e $z \in \mathcal{Z}$, è facile risalire al testo in chiaro x, facendo $D(z, k) = x$ per quell'unica chiave $k \in \mathcal{K}$ che soddisfa $g(s) = k$ (equivalentemente $h(k) = s$). Si noti che questo non significa che è facile calcolare $g(s) = k$ e quindi poi $D(z, k) = x$, poiché l'ultima operazione è difficile (per (**CP3**)); ciò che è facile da fare è la funzione composta D^*.

Torniamo al nostro esempio per capire come funziona il sistema. Alice vuole criptare ed inviare il messaggio x a Bob. Sapendo la chiave pubblica di Bob, che chiamiamo k, Alice calcola senza difficoltà (per (**CP2**)), $E(x, k) = z$ e spedisce z a Bob. Ora, Bob già conosce la sua chiave segreta s, quindi per decrittare il messaggio gli basta eseguire l'operazione $D^*(z, s)$. Infatti, poiché $g(s) = k$, Bob otterrà

$$D^*(z, s) = D(z, g(s)) = D(z, k) = D(E(x, k), k) = x,$$

che è il messaggio originale di Alice.

Veniamo adesso ad Eva. La sua posizione è diversa da quella di Bob per il solo fatto che lei non conosce la chiave segreta s. Eva può fare solo due cose, o calcolare

direttamente $D(z,k)$ (operazione difficile in virtù dell'assioma (**CP3**), oppure decidere di trovare la chiave segreta s di Bob, calcolando $h(k)$ (ma anche questa operazione è difficile per (**CP6**). Si noti che Eva in principio sa come calcolare entrambe le funzioni D ed h; l'unica cosa che rende sicura la procedura è la complessità di queste operazioni. Inoltre la cosa importante è la segretezza della chiave s, che assicura a Bob di poter decodificare il messaggio. Questa chiave scelta da Bob non viene mai comunicata a nessuno e lui se ne serve solo per decrittare i messaggi che gli vengono spediti.

Quindi in principio abbiamo un metodo che consente ad un gruppo, anche esteso, di persone di poter comunicare in modo sicuro. Supponiamo infatti che ci sia un gruppo di utenti: A, B, C,... Per prima cosa, ognuno sceglie una sua chiave segreta: Alice sceglie s_A, Bob s_B, e così via. Poi Alice calcola $k_A = g(s_A)$ e la rende pubblica(in una directory o sulla sua pagina Web). Così fa Bob con $k_B = g(s_B)$ e tutti gli altri a seguire. Ora se uno qualunque vuole mandare un messaggio, diciamo x, ad Alice per prima cosa ottiene la chiave pubblica k_A di Alice, poi critta x come $z = D(x, k_A)$ e lo invia ad Alice. Lei, e solo lei, può agevolmente calcolare x come $x = D^*(z, s_A) = D(z, k_A)$, per tutti gli altri, tale conto è difficile.

Per quanto riguarda la crittografia a chiave pubblica, le funzioni di codifica E sono tutte trapdoor one-way, dove la chiave segreta funge da trapdoor; mentre le funzioni g dall'insieme delle chiavi segrete a quello delle chiavi pubbliche sono tutte funzioni one-way.

Firme digitali

In tutta questa procedura c'è però un potenziale problema. Ovviamente per quanto detto sopra, Eva non può leggere i messaggi che Alice invia a Bob, però, poiché la chiave di Bob è pubblica, Eva potrebbe benissimo scrivere a Bob fingendosi di essere Alice e sostituire così i messaggi di Alice con i suoi. Vediamo come si può la crittografia a chiave pubblica risolve questo inconveniente.

(**CP7**) L'insieme $\mathcal{P}$ dei testi in chiaro coincide con $\mathcal{C}$ insieme dei testi cifrati.

Questo assunto non è assolutamente restrittivo, in quanto, quasi sempre in pratica, i messaggi sono stringhe di bit ed entrambi gli insiemi coincidono quindi con $\{0,1\}^{\mathbb{N}}$.

(**CP8**) $E(D(z,k),k) = z$ per ogni $z \in \mathcal{C}$ e ogni $k \in \mathcal{K}$.

In pratica le funzioni $E(\cdot,k)$ e $D(\cdot,k)$ possono non essere delle biezioni, poiché gli insiemi dei potenziali plaintext può essere infinito, per questo si fa questa assunzione.

C'è un modo semplice con cui Alice può autenticare con sicurezza il messaggio x che intende spedire a Bob. Per prima cosa, fa un'operazione che può sembrare bizzarra, ovvero finge che x sia un testo cifrato e lo decodifica con la sua chiave segreta, cioè esegue $D(x, k_A) =: u$. Poi scrive in chiaro un preambolo del tipo "Questo messaggio è firmato da Alice", ed infine codifica il tutto usando la chiave pubblica di Bob, cioè calcola $z = E(u, k_B) = E(D(x, k_A), k_B)$ che invia a Bob. A questo punto Bob decodifica z, usando la sua chiave segreta, ottenendo $D(z, k_B) = u$. Ai suoi occhi u si presenta come il preambolo "Questo messaggio è firmato da Alice" seguito da un testo incomprensibile. Per leggere effettivamente x, a Bob ora basta prendere questo testo incomprensibile e codificarlo con la chiave pubblica di Alice, ottenendo

$$E(u, k_A) = E(D(x, k_A), k_A) = x.$$

Si noti che Alice è l'unica persona in grado di fare agevolmente $D(x, k_A)$, poiché l'unica che possiede la chiave segreta s_A e $k_A = g(s_A)$, pertanto Bob ha così la certezza che il messaggio proviene a tutti gli effetti da Alice.
Nel Capitolo 5 affronteremo in dettaglio le firme digitali.

4.1 RSA

Nel 1977 tre ricercatori del MIT, Ronald Rivest, Adi Shamir e Leonard Adleman, hanno reso pubblico il primo crittosistema a chiave pubblica ad essere stato efficacemente implementato. L'algoritmo da loro inventato, denominato RSA per via delle iniziali dei loro cognomi, è asimmetrico e si basa sulla difficoltà pratica di fattorizzare un numero intero che sia prodotto di due primi grandi.

Preliminari

Dal punto di vista teorico, il sistema RSA si basa sui seguenti problemi, che classifichiamo in facili e difficili.

Problemi facili

i) Verificare se un intero n sia un numero primo.

ii) Dati a ed n, trovare $MCD(a, n)$ ed eventualmente un inverso di $a \pmod n$.

iii) Calcolare la funzione "potenza e-esima modulo n", $T_e : x \longmapsto x^e \pmod n$.

Problemi difficili

iv) Dato un intero n, trovarne la sua fattorizzazione.

v) Dato un intero n, calcolare $\lambda(n)$ (o equivalentemente $\varphi(n)$).

vi) Dati gli interi n ed e, trovare $d \in \mathbb{N}$ tale che T_d è l'inversa di $T_e \pmod n$.

Per quanto visto nel Capitolo 2 sappiamo che i primi tre problemi appartengono alla classe P, ovvero si risolvono in tempo polinomiale. Per i problemi iv), v) e vi) al momento non sono noti algoritmi efficienti per risolverli, e, con i mezzi attuali, il dispendio in termini di tempo per risolvere uno di questi problemi rende RSA pressoché sicuro. Però per nessuno di questi tre problemi è stata provata la NP-completezza; è quindi possibile che alla fine iv),v) e vi) non siano affatto così difficili come vorremmo (e di conseguenza RSA non sia sicuro come schema crittografico). A tal fine è interessante ricordare che nel 1994 Peter Shor [25] ha trovato un algoritmo quantico (ovvero che opera su eventuali computer quantistici) in grado di risolvere il problema della fattorizzazione (problema i)) in tempo polinomiale, più correttamente tale problema viene detto di tipo $\mathsf{BQP} - \mathsf{Time}$ (per *"Bounded error Quantum Polynomial time"*).

Se assumiamo che n sia il prodotto di due primi distinti, i problemi iv) e v). sono equivalenti. Vale infatti la seguente

Proposizione 4.1. *Sia n il prodotto di due numeri primi distinti. Allora i seguenti problemi sono computazionalmente equivalenti.*

$P_1 =$ *determinare i fattori primi di n,*

$P_2 =$ *determinare $\varphi(n)$,*

$P_3 =$ *determinare $\lambda(n)$.*

Dimostrazione. Abbiamo già provato alla fine del Capitolo 2 che P_1 e P_2 sono equivalenti dal punto di vista computazionale. Proviamo ora che $P_2 \simeq P_3$.

Supponiamo di avere un oracolo per P_2, allora, poiché $P_2 \simeq P_1$, in tempo polinomiale possiamo determinare p e q tali che $n = pq$. Usando l'algoritmo di Euclide si determina in tempo polinomiale $\delta = M.C.D.(p-1, q-1)$ e quindi anche

$$\lambda(n) = m.c.m.(p-1, q-1) = \frac{\varphi(n)}{\delta}.$$

Viceversa, supponiamo di avere un oracolo per P_3. Allora n e $\lambda(n)$ sono noti. Sia q il maggiore dei due primi (incogniti) della fattorizzazione di n. Sappiamo che $\lambda(n)$ è un multiplo di $q-1$ e divide $\varphi(n)$. Chiamiamo r il resto della divisione di n per $\lambda(n)$, allora abbiamo che

$$n - \varphi(n) \equiv r \pmod{\lambda(n)}.$$

D'altra parte

$$n - \varphi(n) = pq - (p-1)(q-1) = p + q - 1 < 2(q-1) < 2\lambda(n)$$

(si assume $n \neq 6$, perché altrimenti è banale). Ne segue che $n - \varphi(n) = r$ oppure $n - \varphi(n) = r + \lambda(n)$, cioè $\varphi(n) = n - r$ oppure $\varphi(n) = n - r - \lambda(n)$. Si noti che tutte le operazioni si eseguono in tempo polinomiale. ■

Implementazione di RSA

Bob sceglie due numeri primi grandi, p_B e q_B (dell'ordine di 2^{1024}), ne fa il prodotto $n_B = p_B q_B$ e calcola $\lambda(n_B) = m.c.m.(p_B - 1, q_B - 1)$. A questo punto sceglie un esponente e_B che sia coprimo con $\lambda(n_B)$. Applica l'algoritmo di Euclide per trovare l'inverso di e_B (mod $\lambda(n_B)$) (ovvero il numero d_B tale che l'applicazione $T_{d_B} : x \longmapsto x^{d_B}$ sia l'inversa di $T_{e_B} : x \longmapsto x^{e_B}$). Da ultimo Bob rende pubblica la chiave $k_B := (n_B, e_B)$ e tiene segreta $s_B := (p_B, q_B, d_B)$. Si noti che $\mathcal{P} = \mathcal{C} = [0, n_B)$, l'insieme degli interi minori di n_B.

Se Alice intende mandare un messaggio a Bob, per prima cosa deve trasformare il suo messaggio in un elemento $x \in U(\mathbb{Z}_{n_B})$. (Per far ciò se il messaggio è una stringa binaria di lunghezza $\geq n_B$, prima lo divide in blocchi ciascuno di lunghezza k, con $2^k < n_B$, poi considera ogni blocco come un intero dell'intervallo $[0, 2^k - 1]$ ovvero un elemento di $\mathbb{Z}_{n_B}$. Poiché

$$\frac{|U(\mathbb{Z}_{n_B})|}{|\mathbb{Z}_{n_B}|} = \frac{\varphi(n_B)}{n_B} = \left(1 - \frac{1}{p_B}\right)\left(1 - \frac{1}{q_B}\right) \approx 1,$$

essendo p_B e q_B grandi, ci sono buone probabilità che ogni blocco sia un elemento invertibile di $\mathbb{Z}_{n_B}$, quindi possa essere codificato e trasmesso da Alice. Se comunque un eventuale blocco non fosse invertibile, una leggera modifica ad esse lo renderebbe tale). A questo punto Alice calcola $z = T_{e_B}(x)$ e invia z a Bob.

Bob decifra il messaggio applicando la funzione inversa T_{d_B}. Ottiene in questo modo

$$T_{d_B}(T_{e_B}(x) = T_{d_B}(x^{e_B}) = (x^{e_B})^{d_B} = x^{e_B d_B}$$

che è un numero minore di n_B e congruo ad x (mod n_B), essendo $e_B d_B \equiv 1$ (mod $\lambda(n_B)$) ed $x^{\lambda(n_B)} = 1$ per ogni $x \in U(\mathbb{Z}_{n_B})$. Poiché $x < n_B$, il risultato è proprio il messaggio x di Alice.

Se Eva intercetta il messaggio z, per decrittarlo dovrebbe calcolare $T_{d_B}(z)$, il che è difficile per lei che non conosce d_B (problema vi) di sopra). Alternativamente, Eva potrebbe trovare d_B dalla chiave pubblica e_B, ma ciò significa trovare l'inverso di e_B modulo $\lambda(n_B)$, e quindi il problema si sposta a trovare $\lambda(n_B)$, che è altrettanto difficile (problema v)). Come ultima istanza, Eva potrebbe cercare la fattorizzazione di n_B. Anche questo però è difficile (problema iv)). In tal modo la cifratura può ritenersi sicura.

La firma digitale

Il sistema RSA è in grado di supportare uno schema di firma digitale.

Alice, che, come Bob e tutti gli utenti del sistema, ha scelto una chiave pubblica e_A ed una privata d_A, prima di inviare il messaggio x a Bob, lo 'decodifica' facendo $T_{d_A}(x)$. Quindi invia a Bob $z := T_{e_B}(T_{d_A}(x))$. Bob, un volta ricevuto z, applica prima T_{d_B} e poi, per verificare l'effettiva autenticità di Alice, applica la funzione T_{e_A}, che è di dominio pubblico. Ottiene in questo modo

$$T_{e_A}(T_{d_B}(z)) = T_{e_A}(T_{d_B}(T_{e_B}(T_{d_A}(x))) = T_{e_A}(T_{d_A}(x)) = x,$$

con la certezza che tale messaggio proviene effettivamente da Alice.[2]

Crittoanalisi di RSA

Presentiamo un attacco al sistema RSA che si divide in due fasi. La prima consiste nel determinare un multiplo m dell'esponente $\lambda(n)$. La seconda fase mostra come dato un tale m si può giungere, con tempistiche pressoché polinomiali, alla fattorizzazione di n. La strategia è dunque quella già esposta nella Proposizione 4.1, ovvero la conoscenza di $\lambda(n)$ implica quella della fattorizzazione di n, in tempo polinomiale. Occorre precisare che le due fasi di questo tipo di attacco si affrontano anche con tecniche diverse da quelle che ora esponiamo.

Prima fase

Vediamo come, sotto opportune ipotesi, si può trovare in tempo polinomiale un multiplo m di $\lambda(n)$. Questo svolgerà un ruolo chiave nella seconda fase dell'attacco.

Ricordiamo lo sviluppo in *frazione continua* di ogni numero reale.

Teorema 4.2. *Ogni numero reale r ammette una scrittura del tipo*

$$r = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{a_4 + \dots}}}}$$

con $a_0 \in \mathbb{Z}$ e $a_i \in \mathbb{N}$ per $i > 0$.

[2] Il lettore attento potrebbe giustamente obiettare che le applicazioni T_{e_A} e T_{d_A} sono biezioni di $U(\mathbb{Z}_{n_A})$ e quindi se $x \in U(\mathbb{Z}_{n_B})$, $T_{d_A}(x)$ non risulta ben definita. In crittografia si ovvia a questo tipo di inconvenienti facendo intervenire una funzione di espansione o compressione (ad esempio le funzioni "hash" della sezione 5.4). A rigore pertanto dovremmo scrivere $T_{d_A}(h(x))$ in luogo di $T_{d_A}(x)$, dove in questo caso h è una particolare funzione $h : U(\mathbb{Z}_{n_B}) \longrightarrow U(\mathbb{Z}_{n_A})$.

Dimostrazione. Si veda ad esempio [14] o [23]. ■

La frazione continua di sopra viene denotata con la scrittura

$$[a_0; a_1, a_2, a_3, a_4, \dots].$$

Una frazione continua si dice finita se è della forma $[a_0; a_1, a_2, \dots, a_n]$ per qualche $n \in \mathbb{N}$, ovvero se coinvolge solo un numero finito di interi. È facile provare la seguente

Proposizione 4.3 ([14])**.** *I numeri rappresentabili tramite una frazione continua finita sono tutti e soli i numeri razionali.*

Poichè $[a_0; a_1, \dots, a_n + 1] = [a_0; a_1, \dots, a_n, 1]$, in genere non c'è unicità di scrittura per i numeri razionali. Se però supponiamo che l'ultimo termine sia necessariamente > 1, allora sì essendo $a_0 = \lfloor \frac{a}{b} \rfloor$, $a_1 = \lfloor (\frac{a}{b} - \lfloor \frac{a}{b} \rfloor)^{-1} \rfloor$, e così via.
Si noti che la lunghezza della scrittura di $\frac{a}{b}$ dipende dal numero di iterazioni dell'algoritmo di Euclide, pertanto ha lo stesso ordine di grandezza di $O(\text{size}\,(b))$. Per quando riguarda invece i numeri irrazionali l'unicità di scrittura in forma di frazione continua è sempre garantita.

Definizione 16. *Sia α un numero reale e sia $\alpha = [a_0; a_1, a_2, \dots]$ la sua frazione continua. Un* convergente ad α *è per definizione un troncamento $[a_0; a_1, a_2, \dots, a_k]$ di α.*

Si noti che i convergenti di un razionale $\frac{a}{b}$ sono tanti quanti i passaggi necessari per determinare (a, b), ovvero sono un $O(\text{size}\,(b))$.

Teorema 4.4 ([14])**.** *Sia $\alpha \in \mathbb{R}$. Se $\left|\alpha - \frac{a}{b}\right| < \frac{1}{b^2}$, allora $\frac{a}{b}$ è un convergente ad α.*

Teorema 4.5. *Siano p e q due numeri primi distinti tali che $p < q < 2p$. Sia $n = pq$ e siano $e, d \in [0, \varphi(n))$ tali che $ed \equiv 1$ (mod $\varphi(n)$). Se*

$$d < \frac{\sqrt[4]{n}}{\sqrt{6}}$$

allora è possibile determinare d in tempo polinomiale conoscendo solamente n ed e.

Dimostrazione. Dalle ipotesi segue che

$$n - \varphi(n) = p + q - 1 < 3p < 3\sqrt{n}. \tag{4.1}$$

Sia ora $k \in \mathbb{Z}$ un numero coprimo con d tale che $ed - k\varphi(n) = 1$ (sappiamo che esiste poiché $ed \equiv 1 \pmod{\varphi(n)}$). In particolare si ha che $(d, k) = 1$. Valutiamo ora

$$\begin{aligned}\left|\frac{e}{n} - \frac{k}{d}\right| &= \frac{|ed - k(n - \varphi(n) + \varphi(n))|}{nd} = \frac{|ed - k\varphi(n) - k(n - \varphi(n))|}{nd} \\ &= \frac{|1 - k(n - \varphi(n))|}{nd} < \frac{k(n - \varphi(n))}{nd} < \frac{3k}{d\sqrt{n}}.\end{aligned}$$

Essendo

$$0 < k = \frac{ed-1}{\varphi(n)} < \frac{e}{\varphi(n)}d < d,$$

si ottiene

$$\left|\frac{e}{n} - \frac{k}{d}\right| < \frac{3}{\sqrt{n}}. \tag{4.2}$$

Ora per ipotesi abbiamo che $d < \frac{\sqrt[4]{n}}{\sqrt{6}}$, da cui segue $\frac{1}{6d^2} > \frac{1}{\sqrt{n}}$. Pertanto da (4.2) si deduce che

$$\left|\frac{e}{n} - \frac{k}{d}\right| < \frac{1}{2d^2}.$$

Per il Teorema 4.4 abbiamo che $\frac{k}{d}$ è un convergente ad $\frac{e}{n}$. Ora poiché n ed e sono pubblici possiamo stilare in tempo polinomiale (usando l'algoritmo di Euclide) una lista di tutti i convergenti ad $\frac{e}{n}$

$$\frac{x_1}{y_1}, \frac{x_2}{y_2}, \ldots, \frac{x_t}{y_t}, \qquad \text{con } (x_i, y_i) = 1.$$

Essendo $(k, d) = 1$, la frazione $\frac{k}{d}$ è ridotta, dunque $d \in \{y_1, y_2, \ldots, y_t\}$. Per trovare l'esatto valore di d, in tempo polinomiale, si può prendere un messaggio qualunque $\alpha \in U(\mathbb{Z}_{n_B})$, calcolare $\beta = \alpha^e$ e fare β^{y_i} (per $i \in \{1, 2, \ldots, t\}$) fino a quando $\beta^{y_j} = \alpha$ per qualche j, allora si avrà che $d = y_j$. ■

Osservazione 5. Se Bob sceglie i primi p_B, q_B in modo tale che $p_B < q_B < 2p_B$ e come chiave segreta $d_B < \frac{\sqrt[4]{n_B}}{\sqrt{6}}$, con $e_B d_B \equiv 1 \pmod{\varphi(n_B)}$ si ha che le ipotesi del Teorema 4.5 sono tutte soddisfatte. Eva pertanto è in grado di determinare in tempo polinomiale d_B e quindi di leggere i messaggi criptati che vengono inviati a Bob. Inoltre, Eva ottiene anche un multiplo $m = e_B d_B - 1$ di $\varphi(n_B)$ che sarà utile per la seconda fase dell'attacco. Vedremo infatti che anche se Bob decidesse di cambiare le chiavi e_B, d_B, Eva potrebbe comunque agevolmente rompere la sicurezza del sistema. A Bob non rimane altro da fare che ripartire con la scelta di due nuovi numeri primi.

Seconda fase

Questa fase dell'attacco ad RSA utilizza la seguente Proposizione (si veda l'analogia con la Proposizione 4.1).

Proposizione 4.6. *Sia n un intero naturale che è il prodotto di due numeri primi distinti non noti. Supponiamo di aver trovato un intero m multiplo di $\varphi(n)$ ed un elemento $\alpha \in U(\mathbb{Z}_n)$ tale che $\alpha^{\frac{m}{2}} \neq \pm 1$. Allora la fattorizzazione di n si può trovare in tempo polinomiale.*

Dimostrazione. Detto $\alpha = a+n\mathbb{Z}$, con $a \in \{1, n-1\}$, dal fatto che $\alpha^m = 1$, segue che $n|a^m - 1 = (a^{\frac{m}{2}} + 1)(a^{\frac{m}{2}} - 1)$. Inoltre poiché $\alpha^{\frac{m}{2}} \neq \pm 1$, abbiamo che $n \not| a^{\frac{m}{2}} + 1$ e $n \not| a^{\frac{m}{2}} - 1$, e quindi i due fattori primi che compongono n sono fattori rispettivamente di $a^{\frac{m}{2}} + 1$ e di $a^{\frac{m}{2}} - 1$. A questo punto basta applicare l'algoritmo di Euclide (che richiede un tempo polinomiale) per determinare $(n, a^{\frac{m}{2}} + 1)$ e $(n, a^{\frac{m}{2}} - 1)$. ■

Vediamo ora come procede l'algoritmo per determinare m ed α come nelle ipotesi della Proposizione 4.6. Per prima cosa supponiamo di avere m multiplo di $\varphi(n)$, ovvero tale che per ogni $\alpha \in U(\mathbb{Z}_n)$ si abbia $\alpha^m = 1$. Un tale m può essere stato trovato ad esempio utilizzando la strategia esposta nella prima fase. Poiché $\varphi(n)$ è pari, anche m lo 'e. Cerchiamo a questo punto un elemento $\alpha \in U(\mathbb{Z}_n)$ tale che $\alpha^{\frac{m}{2}} \neq 1$. La ricerca di un tale α si può condurre in modo non esaustivo, sfruttando la seguente

Proposizione 4.7. *Siano n ed m come sopra e sia*

$$\mathcal{A}_m = \left\{ \alpha \in U(\mathbb{Z}_n) | \alpha^{\frac{m}{2}} \neq 1 \right\}.$$

Se $\mathcal{A}_m \neq \varnothing$, allora $|\mathcal{A}_m| \geq \frac{\varphi(n)}{2}$ e la probabilità che scegliendo a caso un elemento β di $U(\mathbb{Z}_n)$ questo sia tale che $\beta^{\frac{m}{2}} = 1$ è minore o uguale ad $1/2$.

Dimostrazione. Sia $\alpha \in \mathcal{A}_m$ e definiamo l'applicazione

$$\begin{aligned} f : U(\mathbb{Z}_n) \setminus \mathcal{A}_m &\longrightarrow \mathcal{A}_m \\ \gamma &\longmapsto \gamma\alpha \end{aligned}$$

È immediato verificare che f è ben definita ed iniettiva. Ne segue che $|\mathcal{A}_m| \geq \frac{\varphi(n)}{2}$ e quindi la probabilità che un arbitrario elemento invertibile di $\mathbb{Z}_n$ non stia in $\mathcal{A}_m$ è

$$\frac{|U(\mathbb{Z}_n) \setminus \mathcal{A}_m|}{|U(\mathbb{Z}_n)|} = 1 - \frac{\varphi(n)}{|\mathcal{A}_m|} \leq \frac{1}{2}.$$

□

Possiamo ora impostare il seguente algoritmo.

1. Scegli $\alpha \in \mathbb{Z}_n$, $\alpha \neq 0$, $\alpha = a + n\mathbb{Z}$.

2. Calcola $d = (a, n)$. Se $d \neq 1$ allora d è uno dei due fattori di n e siamo in grado di fattorizzare n. Altrimenti $\alpha \in U(\mathbb{Z}_n)$.

3. Calcola $\alpha^{\frac{m}{2}}$. Se $\alpha^{\frac{m}{2}} \neq 1$, esci. Altrimenti vai al passo 1.

Pensando di eseguire k cicli al nostro algoritmo, la probabilità di trovare k elementi nessuno dei quali appartenente ad $\mathcal{A}_m$ è $\leq \frac{1}{2^k}$, che è molto bassa già per $k = 20$ (parametro per cui il costo computazionale dell'algoritmo non crea difficoltà). Allora è ragionevole pensare che se dopo k cicli ancora non si è trovato un elemento di $\mathcal{A}_m$, questo insieme sarà vuoto e pertanto sarà possibile rimpiazzare m con la sua metà. Si procede in questo modo, continuando eventualmente a dimezzare m, alla peggio ci si arresta quando m sarà dispari. L'ordine di grandezza dell'algoritmo è quindi un $O(\text{size}\,(m))$.

Pertanto abbiamo provato che: assegnato un multiplo di $\varphi(n)$, in tempo polinomiale siamo in grado di trovare un eventuale altro multiplo m di $\varphi(n)$, ed un $\alpha \in U(\mathbb{Z}_n)$ per cui $\alpha^{\frac{m}{2}} \neq 1$. Mostriamo ora che un tale α ha esattamente una probabilità del 50% di essere tale che $\alpha^{\frac{m}{2}} \neq \pm 1$.
Per il Teorema cinese dei resti (Teorema 1.3) $\mathbb{Z}_n \simeq \mathbb{Z}_p \times \mathbb{Z}_q$ (dove p e q sono i due fattori primi non noti di $n = pq$). Rappresentiamo pertanto l'elemento α come la coppia ordinata (ϵ, η), con $\epsilon \in \mathbb{Z}_p^*$ ed $\eta \in \mathbb{Z}_q^*$. Dal fatto che $\varphi(n)|m$ e $\alpha^{\frac{m}{2}} \neq 1$ segue che il minimo comune multiplo tra $p-1$ e $q-1$, $\lambda(n)$, divide m e non divide $m/2$. In particolare si ha che

$$\alpha^{\frac{m}{2}} = (\epsilon^{\frac{m}{2}}, \eta^{\frac{m}{2}}) \in \{(1,-1), (-1,1), (-1,-1) = -1\},$$

infatti se r è un primo, l un multiplo di r e $x \in \mathbb{Z}_r$ si ha che $x^{\frac{l}{2}} = -1$ se e solo se x è un non quadrato di $\mathbb{Z}_r$, altrimenti $x^{\frac{l}{2}} = 1$ (lo si dimostri per esercizio).
Si possono presentare due casi.

i) $m/2$ è multiplo di uno solo fra $p-1$ e $q-1$.

ii) $m/2$ non è multiplo né di $p-1$ né di $q-1$.

Nel primo caso, supponendo ad esempio che $p-1|\frac{m}{2}$ e $q-1 \not| \frac{m}{2}$ (la situazione opposta si tratta in modo analogo), abbiamo che

$$\alpha^{\frac{m}{2}} = (\epsilon^{\frac{m}{2}}, \eta^{\frac{m}{2}}) = (1,-1).$$

Notiamo che mentre ϵ può essere scelto arbitrariamente in $\mathbb{Z}_p^*$, l'elemento η deve necessariamente essere un non-quadrato di $\mathbb{Z}_q^*$. Il numero di tali α è pertanto dato da

$$(p-1) \cdot \frac{q-1}{2} = \frac{\varphi(n)}{2}.$$

Similmente si ottiene lo stesso numero se $q-1|m/2$ e $p-1 \not| m/2$.
Nel secondo caso, le coppie (ϵ, η) per cui

$$(\epsilon^{\frac{m}{2}}, \eta^{\frac{m}{2}}) \neq \pm(1,1)$$

si ottengono esattamente quando uno solo fra ϵ ed η è un quadrato. Il numero delle scelte possibili è allora dato da

$$\frac{p-1}{2} \cdot \frac{q-1}{2} + \frac{p-1}{2} \cdot \frac{q-1}{2} = \frac{(p-1)(q-1)}{2} = \frac{\varphi(n)}{2}.$$

In entrambi i casi, i) e ii), abbiamo quindi il 50% di probabilità che $\alpha^{\frac{m}{2}} \neq \pm 1$ e possiamo ora concludere utilizzando la Proposizione 4.6.

4.1.1 Osservazioni sulla scelta dei parametri

1. La scelta dei primi p e q grandi è giustificata senz'altro dal fatto che vogliamo sia molto difficile per un attaccante trovare la fattorizzazione di $n = pq$. Inoltre, come già osservavamo, è importante che scegliendo a caso un elemento di $\mathbb{Z}_n$, questo sia, con grande probabilità, un elemento invertibile. Ora, questa probabilità è data da

$$\frac{\varphi(n)}{n} = \left(1 - \frac{1}{p}\right)\left(1 - \frac{1}{q}\right)$$

   che per l'appunto è prossima ad 1 se p e q sono numeri grandi.

2. Nella scelta di p e q, bisogna stare attenti che non ricadano in sottoclassi particolari di numeri primi, facilmente calcolabili. Ad esempio p e q non devono essere primi di Fermat (della forma $2^k + 1$), né di Mersenne ($2^k - 1$), né di Pierpont ($2^h 3^k + 1$) o simili.

3. p e q non devono essere troppo vicini. Il Teorema 4.5 impone che il maggiore dei due non sia minore del doppio dell'altro.

4. In generale vale che se i primi p e q sono scelti in modo tale che tra i fattori comuni di $p-1$ e $q-1$ ci siano primi grandi e con esponente basso, allora la ricerca di un multiplo m di $\lambda(pq)$ risulta difficile.

5. Anche la scelta della chiave pubblica e deve essere ponderata. Da un lato e piccolo agevola l'operazione di codifica, dall'altro pone lo schema a rischio del cosiddetto "low exponent attack". Supponiamo che un mittente, ad esempio una banca, intende spedire lo stesso messaggio x ai suoi clienti. Supponiamo che ci siano almeno e clienti, i cui moduli indicheremo con $n_1, n_2, \ldots, n_e$, che abbiano scelto lo stesso esponente e come chiave pubblica; si noti che più e è piccolo più questa assunzione risulta credibile. Allora la banca invierà i messaggi $z_i = x^e \pmod{n_i}$ per $i = 1, 2, \ldots, e$. Un attaccante che intercetta tutti gli z_i, tramite il Teorema cinese dei resti (Teorema 1.11), risale ad un intero $z \equiv x^e \pmod{n_1 n_2 \ldots n_e}$. Ora

poiché $x < n_i$ per ogni i, abbiamo che $x^e < n_1 n_2 \dots n_e$. Dunque $z = x^e$ e adesso x può essere ricavato semplicemente estrapolando la radice e-esima di z.

6. L'esponente e deve anche essere scelto in modo che il suo inverso d, modulo $\lambda(n)$, sia non tropppo piccolo. Abbiamo visto infatti nel Teorema 4.5 che d piccolo espone il nostro schema ad un attacco di I fase. Inoltre, nel 2000 D. Boneh e G. Durfee in [6] hanno provato che in generale se $d < n^{0,292}$ allora RSA si rompe sempre.

4.2 Il logaritmo discreto

Sia G un gruppo ciclico finito di ordine n e sia g un generatore di G. Preso un qualunque $h \in G$ esisterà un unico intero $x \in \{0, 1, \dots, n-1\}$ tale che $h = g^x$. In analogia con l'Analisi reale, l'intero x si dice *logaritmo discreto di h in base g* e viene indicato col simbolo $x = \log_g(h)$.

L'importanza del logaritmo discreto in Crittografia è dovuta al semplice fatto che, mentre, come abbiamo visto nel Capitolo 2, calcolare le potenze di un elemento è un'operazione in P, altrettanto non si può dire per il calcolo del logaritmo discreto. Fino ad oggi infatti non sono noti metodi efficienti per tale computo. Questo fa sì che l'operazione di elevamento a potenza, all'interno di ogni gruppo finito, sia una funzione one-way. I protocolli che esporremo di seguito si basano tutti su questo problema del logaritmo discreto (in sintesi "DLP", per *Discrete Logarithm Problem*).

4.2.1 Il protocollo Diffie-Hellman-Merkle

Il protocollo di Diffie-Hellman-Merkle (in sintesi: DHM[3]) del 1976 è storicamente considerato una delle pietre miliari della crittografia a chiave pubblica. Rappresenta infatti il primo esempio di sistema crittografico asimmetrico. In realtà non si tratta a tutti gli effetti di un crittosistema a chiave pubblica, per come questi sono stati definiti all'inizio del Capitolo, comunque funge da base per diversi protocolli a chiave pubblica (uno fra tutti il sistema ElGamal). DHM permette a due utenti qualsiasi, per noi rappresentati dai soliti Alice e Bob, di scambiarsi chiavi segrete, utilizzando canali che di per sé sono insicuri. Vediamo in dettaglio come funziona.

Supponiamo che Alice e Bob vogliano comunicare tramite un sistema crittografico simmetrico, ma abbiano problemi ad incontrarsi per scambiarsi le chiavi. Per prima cosa si accordano sulla scelta di un gruppo ciclico G di ordine finito, diciamo n, e di un suo generatore g. Il gruppo G e l'elemento g vengono resi pubblici. Dopodiché Alice sceglie

[3] Il nome comunemente usato per questo protocollo è Diffie-Hellman. Fu Hellman nel 2002 a suggerire che l'algoritmo venisse giustamente chiamato Diffie-Hellman-Merkle per riconoscere il contributo di Ralph Merkle all'invenzione (si veda [20]).

un intero $a \in \{1, \ldots, n-1\}$ e invia g^a a Bob, stando per attenta a non rivelare l'intero a. Bob a sua volta sceglie un intero $b \in \{1, \ldots, n-1\}$, calcola g^b che invia ad Alice, mantenendo b segreto. Entrambi ora possono tranquillamente calcolare g^{ab}, infatti essendo $g^{ab} = (g^b)^a$, a Alice basta elevare alla potenza a (che lei sola conosce) il messaggio di Bob, g^b, similmente, essendo $g^{ab} = (g^a)^b$, Bob eleva il messaggio ricevuto g^a alla potenza b (che lui solo conosce). Si noti però che un attaccante, Eva, conosce solo g, g^a e g^b, quindi per poter risalire a g^{ab} e forzare il sistema Eva è costretta a risolvere un'istanza del DLP.

Le differenze col sistema RSA sono sostanzialmente due. Innanzitutto DHM permette solo ad una coppia di utenti per volta di comunicare in maniera sicura, invece RSA gestisce la comunicazione sicura fra coppie di un'intera rete. In secondo luogo, per poter generare la chiave segreta, questo protocollo necessita che vengano scambiati almeno tre messaggi, mentre per RSA ne basta uno solo.

4.2.2 Il protocollo ElGamal

Il protocollo ElGamal è un sistema a chiave pubblica che si basa sul protocollo di DHM e quindi sul problema del logaritmo discreto.

Bob sceglie un gruppo ciclico finito G, di ordine diciamo n, ed un suo generatore g. Poi sceglie un intero $b \in \{1, \ldots, n-1\}$, calcola g^b, rende pubblici (g, g^b) e tiene segreto b. Alice, che vuole mandare un messaggio segreto a Bob, per prima cosa lo codifica come un elemento di G, che chiameremo x. Quindi determina anche lei un intero $a \in \{1, \ldots, n-1\}$, calcola la coppia $(\alpha, \beta) = (g^a, x(g^b)^a)$ che invia a Bob. Si noti quindi che il testo cifrato è la coppia (α, β), lunga possiamo dire il doppio del plaintext x. Bob riceve il messaggio (α, β) e conoscendo la sua chiave privata b, risale semplicemente al messaggio tramite

$$\beta(\alpha)^{-b} = xg^{ab}(g^b)^{-a} = xg^{ab}g^{-ab} = x.$$

Si noti che Bob risale al messaggio in chiaro senza conoscere la chiave segreta a di Alice. Eva, che intercetta il messaggio di Alice, deve affrontare uno dei seguenti problemi: o trovare b, la chiave segreta di Bob, in modo da risalire ad x esattamente come fa Bob, oppure trovare a, la chiave segreta di Alice, e risalire ad x calcolando $\beta \cdot (g^b)^{-a}$. Entrambe le strategie di attacco coinvolgono il DLP, e quindi possono essere considerate difficili da fare.

4.2.3 Il protocollo di Massey-Omura

Supponiamo che Alice voglia inviare un oggetto personale a Bob. Allora prende l'oggetto, lo mette in una scatola che protegge con un lucchetto A, che lei sola è in grado di aprire. Bob appena ricevuta la scatola, pone anch'egli un lucchetto, che chiamiamo B, che solo

lui è in grado di aprire. Rispedisce poi la scatola con i due lucchetti ad Alice. Questa ora toglie il lucchetto A e invia per una seconda volta la scatola a Bob, il quale possedendo le chiavi del lucchetto B può finalmente aprire la scatola e vederne il contenuto. Si noti che in tutti e tre le spedizioni la scatola rimane sempre allucchettata e quindi il suo contenuto è (idealmente) protetto. Inoltre Alice e Bob non hanno mai dovuto scambiarsi fisicamente le chiavi. Vediamo ora come questa idea viene implementata nel protocollo in questione.

Il sistema originale ha come plaintext e ciphertex il gruppo moltiplicativo di un campo finito, nulla però ci vieta di essere più generali e lavorare in un qualsiasi gruppo ciclico finito $G = \langle g \rangle$ d'ordine n. Alice e Bob scelgono ciascuno rispettivamente due interi (x_A, y_A) e (x_B, y_B) tali che $x_A y_A \equiv 1 \pmod{n}$ e $x_B y_B \equiv 1 \pmod{n}$. Poi Alice prende il messaggio m, che pensiamo codificato come un elemento di G, e lo codifica calcolando $z = m^{x_A}$. Quindi invia z a Bob. Egli calcola

$$u = z^{x_B} = (m^{x_A})^{x_B} = m^{x_A x_B}$$

che rispedisce subito ad Alice. A questo punto Alice toglie il suo lucchetto facendo $v = u^{y_A} = (m^{x_A x_B})^{y_A} = m^{x_A x_B y_A} = m^{x_A y_A x_B} = m^{x_B}$, dal momento che $x_A y_A \equiv 1 \pmod{n}$, ed invia nuovamente il risultato a Bob. Ora a Bob non basta che togliere il suo lucchetto e leggere il contenuto

$$v^{y_B} = (m^{x_B})^{y_B} = m^{x_B y_B} = m.$$

Notiamo che se Eva intende attaccare il sistema ha a disposizione m^{x_A}, m^{x_B} ed $m^{x_A x_B}$. Non esistono tecniche che permettano di risalire ad m con solo questi dati, senza risolvere istanze del logaritmo discreto. Nonostante ciò il protocollo di Massey-Omura, così come è appena stato presentato, è suscettibile ad un tipo di attacco molto semplice, detto *Man in the middle*. Non appena Alice manda il primo messaggio a Bob, Eva lo intercetta e, fingendo di essere lei stessa Bob, implementa un Massey-Omura con Alice. Alice, pensando di scrivere a Bob, finisce in realtà per inviare poi il messaggio ad Eva, questa poi, dopo averlo letto, potrebbe anche far arrivare il messaggio di Alice a Bob ed implementare uno scambio anche con lui. Siamo nella situazione in cui Alice e Bob pensano entrambi di dialogare in totale segretezza, quando in realtà Eva sta leggendo tutti i messaggi. Questo tipo di attacco fa sì che il protocollo Massey-Omura sia completamente inutile senza un efficace sistema di firma digitale, che vedremo in dettaglio nel Capitolo 5.

4.2.4 Algoritmi per il calcolo del logaritmo discreto

In questa sezione approfondiremo più in dettaglio le difficoltà che sorgono nel risolvere il logaritmo discreto. Come abbiamo visto molti protocolli a chiave pubblica basano la loro sicurezza su questo problema. Presentiamo alcuni metodi di attacco tra i più noti.

Per fissare la notazione, sia G un gruppo ciclico finito d'ordine n e sia g un suo generatore. Assegnato un elemento arbitrario $h \in G$ si vuole determinare l'intero $x \in \{0, 1, \ldots, n-1\}$ tale che

$$h = g^x. \tag{4.3}$$

La ricerca esaustiva

Il primo metodo che salta in mente per determinare il logaritmo discreto x dell'equazione (4.3) è la ricerca esaustiva, cioè testare se $x = 0, 1, 2, \ldots$ soddisfa l'equazione. Non appena si ha una risposta affermativa il logaritmo discreto è trovato. Questo metodo richiede $x - 1$ moltiplicazioni, x confronti e solo 3 dati da memorizzare. Poiché nelle applicazioni crittografiche si ha sempre $x > 2^{160}$, il numero di o.e. è molto grande. Infatti il numero massimo di o.e. è dell'ordine di $O(n) = O(\mathrm{e}^{\log_2(n)})$, mostrando che la complessità è esponenziale. La ricerca esaustiva è pertanto improponibile.

L'algoritmo "Baby-Step & Giant-Step"

Questo algoritmo ideato da D. Shanks ([24]) richiede un numero inferiore di o.e., a discapito di un maggior numero di dati da memorizzare. Descriviamo l'algoritmo.

Poniamo $m = \lceil n \rceil = \lfloor n \rfloor + 1$. Il logaritmo discreto x viene univocamente determinato una volta che vengono trovati q ed r, rispettivamente quoziente e resto della divisione di x per m. Dal fatto che

$$h = g^x = g^{mq+r},$$

segue che

$$hg^{-r} = (g^m)^q.$$

Il primo passo dell'algoritmo procede nel calcolare e memorizzare il seguente insieme, che viene detto "insieme dei baby-steps"

$$B = \left\{ (hg^{-r}, r) | 0 \leq r \leq m-1 \right\}.$$

Si noti subito che se B contiene la coppia $(1, r)$ allora abbiamo trovato il logaritmo discreto di h, in quanto confrontando le prime componenti abbiamo $hg^{-r} = 1$, da cui segue $h = g^r$, cioè $x = r$. Se invece $(1, r) \notin B$, allora calcoliamo

$$k = g^m.$$

A questo punto testiamo se B contiene una coppia del tipo (k^q, r), per $q = 1, 2, 3, \ldots$. Se così fosse avremmo che

$$hg^{-r} = k^q = g^{mq},$$

ovvero

$$h = g^{mq+r}$$

e il logaritmo discreto $x = mq + r$ è trovato. Gli elementi k^q vengono chiamati "giant-steps". L'algoritmo pertanto procede nel comparare ogni giant-step con tutte le prime componenti dell'insieme B dei baby-steps. Si noti che l'algoritmo ha sempre termine, nel senso che prima o poi trova q ed r giusti. Infatti dal fatto che $0 \leq x \leq n-1$ e $0 \leq r \leq m-1$ si ha che $mq \leq mq + r = x \leq n-1 \leq m^2$, quindi $q \leq m$, ovvero dopo al più m passi, l'elemento (k^q, r) apparterrà a B.

Teorema 4.8. *L'algoritmo "baby-step giant-step" richiede $O(\sqrt{|G|})$ o.e. e necessita una memoria per $O(\sqrt{|G|})$ elementi di G.*

Dimostrazione. Poiché $hg^{-r} = hg^{-r+1}g^{-1}$, ogni baby step necessita solo di moltiplicare g^{-1} per l'elemento ottenuto al passo precedente, in totale quindi si eseguono al massimo m o.e.. Allo stesso modo, dal fatto che $g^{mq} = g^{(q-1)m}g^m$, ogni giant-step si ottiene moltiplicando g^m per il passo precedente. Quindi si eseguono al massimo $2m + 1 = O(\sqrt{n}) = O(\sqrt{|G|})$ o.e.. ■

Se $|G| > 2^{160}$ anche questo algoritmo per il calcolo del logaritmo discreto è impraticabile con i software oggi (Aprile 2015) in uso.

L'algoritmo di Hellman-Pholig-Silver

È un algoritmo molto più sofisticato dei precedenti e riconduce il problema del logaritmo discreto per un gruppo ciclico finito a quello per i gruppi ciclici d'ordine potenza di un primo. Supponiamo infatti di conoscere la fattorizzazione per $|G| = n$ (per gli algoritmi di fattorizzazione rimandiamo il lettore al Capitolo 7)

$$n = \prod_{p|n} p^{e(p)},$$

dove con $e(p)$ indichiamo la massima potenza del primo p che divide n. Lo scopo dell'algoritmo è quello di determinare per ogni $p|n$, un $x_p \equiv x \pmod{p^{e(p)}}$. Una volta trovati tutti questi x_p, al variare di p tra i divisori primi di n, il Teorema Cinese dei Resti (1.11) permette dei determinare in tempo polinomiale un $x \equiv x_p \pmod{p^{e(p)}}$ per ogni p e quindi la soluzione cercata.

Procediamo quindi con l'analisi locale e sia p un divisore primo di n. Poniamo

$$n_p = \frac{n}{p^{e(p)}}, \quad g_p = g^{n_p}, \quad h_p = h^{n_p}.$$

Si noti che l'ordine di g_p è esattamente $p^{e(p)}$ e che

$$g_p^x = h_p.$$

L'elemento h_p appartiene al gruppo ciclico $\langle g_p \rangle$ il cui ordine è $p^{e(p)}$. Pertanto esiste il logaritmo discreto $x(p)$ di h_p in base g_p. Il seguente Teorema mostra come conoscendo ogni $x(p)$, per $p|n$, si può risalire al logaritmo discreto x.

Proposizione 4.9. *Con le notazioni di sopra, per ogni divisore primo p di n, sia $x(p)$ il logaritmo discreto di h_p in base g_p. Sia inoltre $x \in \{0, 1, \ldots, n-1\}$ la soluzione del sistema di congruenze $x \equiv x(p)$ (mod $p^{e(p)}$) per ogni $p|n$. Allora x è il logaritmo discreto di h in base g.*

Dimostrazione. Per ogni $p|n$ si ha che

$$(g^{-x}h)^{n_p} = g_p^{-x}h_p = g_p^{-x(p)}h_p = 1.$$

Pertanto l'ordine di $g^{-x}h$ è un divisore di n_p, per ogni p divisore primo di n, e quindi del loro M.C.D. Poiché gli n_p sono tutti coprimi tra loro, ne segue che $g^{-x}h$ ha ordine uno, ovvero è l'unità del gruppo. ■

Assumiamo pertanto ora che $G = \langle g \rangle$ sia un p-gruppo ciclico, ovvero $|G| = p^e$, per qualche intero positivo e. Vogliamo risolvere l'equazione (4.3)

$$h = g^x,$$

dove ricordiamo che h è preso arbitrariamente in G. Sappiamo che $x < p^e$. Scriviamo x in base p come

$$x = \sum_{i=0}^{e-1} x_i p^i,$$

con $0 \leq x_i < p$, per ogni $0 \leq i \leq e-1$. Mostriamo che ogni coefficiente x_i è un logaritmo discreto in un gruppo ciclico d'ordine p. Infatti se eleviamo alla potenza p^{e-1} l'equazione (4.3) otteniamo

$$h^{p^{e-1}} = g^{p^{e-1}x} \tag{4.4}$$

e poiché

$$p^{e-1}x = x_0 p^{e-1} + p^e(x_1 + x_2 p + \ldots x_{e-1}p^{e-2}) \equiv x_0 p^{e-1} \pmod{p^e}$$

per il piccolo teorema do Fermat (1.1) abbiamo che

$$h^{p^{e-1}} = (g^{p^{e-1}})^{x_0}, \tag{4.5}$$

ovvero il coefficiente x_0 è un logaritmo discreto nel gruppo $\langle g^{p^{e-1}} \rangle$ d'ordine p. I restanti coefficienti vengono ora determinati ricorsivamente. Se supponiamo di conoscere $x_0, x_1, \ldots, x_{i-1}$ allora abbiamo che

$$hg^{-(x_0 + x_1 p + \ldots + x_{i-1}p^{i-1})} = g^{x_i p^i + \ldots + x_{e-1}p^{e-1}}.$$

Ponendo $h_i = hg^{-(x_0+x_1p+\ldots+x_{i-1}p^{i-1})}$ ed elevando alla potenza p^{e-i-1} si ottiene

$$h_i^{p^{e-i-1}} = (g^{p^{e-i}})^{x_i}, \tag{4.6}$$

ovvero x_i è soluzione di un DLP.

Riassumendo, l'algoritmo di Pohlig-Hellman procede in questo modo.
Per prima cosa, dati n, g ed h, vengono calcolati $g_p = g^{n_p}$ e $h_p = h^{n_p}$, per ogni divisore primo p di n. Poi si determinano i coefficienti $x_i(p)$ per ogni primo $p|n$ ed ogni $0 \leq i \leq e(p) - 1$, ad esempio usando l'algoritmo "baby-step giant-step". Da ultimo si usa il Teorema cinese dei resti per determinare il logaritmo discreto. La complessità dell'intero procedimento è così stimata

Teorema 4.10. *L'algoritmo di Pohlig-Hellman per il calcolo del logaritmo discreto nel gruppo ciclico G d'ordine n necessita di $O\big(\sum_{p|n} e(p)(\log_2 n + \sqrt{p})\big)$ operazioni gruppali.*

Dimostrazione. Con le stesse notazioni di sopra, il calcolo delle potenze g_p e h_p, per ogni divisore primo p di n, richiede $O(\log_2 n)$ operazioni gruppali. Il calcolo di ogni coefficiente di $x(p)$ in base p ne richiede $O(\log_2 n)$ per eseguire le potenze e $O(\sqrt{p})$ per eseguire il ciclo "baby-step giant-step". Il numero delle cifre di $x(p)$ è $e(p)$. Quindi il costo, in termine di operazioni gruppali, per ogni primo p è dell'ordine $O\big(\sum_{p|n} e(p)(\log_2 n + \sqrt{p})\big)$. Il Teorema cinese dei resti non compie operazioni gruppali. ■

Per il Teorema 2.7 il costo computazione del Teorema cinese dei resti è di $O((\log_2 n)^2)$ o.e. Il Teorema appena provato mostra che il tempo necessario per calcolare il logaritmo discreto con l'algoritmo di Pohlig-Hellman è sostanzialmente limitato da $\sqrt{p}$, dove p è il più grande divisore primo di n. Pertanto se questo primo è considerevolmente piccolo, allora il DLP per G è facile da risolvere.

Esempio 5. Il numero $p = 2 \cdot 3 \cdot 5^{278} + 1$ è primo. Preso $G = GF(p)^*$ il gruppo moltiplicativo del campo d'ordine p, abbiamo che $|G| = p - 1 = 2 \cdot 3 \cdot 5^{278}$, che è un numero di 649 cifre binarie. Il calcolo del logaritmo discreto in questo gruppo è però molto facile da risolvere poiché 5 è il più grande primo che divide $|G|$.

Capitolo 5

Firme digitali

Definizione 17. *Uno* schema di firma digitale *è una quintupla del tipo*

$$(\mathcal{P}, \mathcal{F}, \mathcal{K}, \mathcal{S}, \mathcal{V})$$

dove le seguenti condizioni sono soddisfatte:

1. *$\mathcal{P}$ è un insieme finito dei possibili messaggi,*
2. *$\mathcal{F}$ è un insieme finito delle possibili firme,*
3. *$\mathcal{K}$, lo spazio delle chiavi, è un insieme finito delle possibili chiavi,*
4. *Per ogni $k \in \mathcal{K}$ sono definite una funzione di firma $\mathrm{sig}_k \in \mathcal{S}$ e la corrispondente funzione di verifica $\mathrm{ver}_k \in \mathcal{V}$, dove*

$$\mathrm{sig}_k : \mathcal{P} \longrightarrow \mathcal{F} \quad e \quad \mathrm{ver}_k : \mathcal{P} \times \mathcal{F} \longrightarrow \{0, 1\}$$

sono tali che per ogni $x \in \mathcal{P}$ ed ogni $f \in \mathcal{F}$

$$\mathrm{ver}_k(x, f) = \begin{cases} 1 & \text{se} \quad \mathrm{sig}_k(x) = f \\ 0 & \text{altrimenti.} \end{cases}$$

Uno schema di firma digitale consiste quindi di due componenti: un algoritmo per la firma ed uno per la verifica. Alice può firmare i propri messaggi x, usando una chiave segreta k e calcolando $\mathrm{sig}_k(x)$. La firma può essere poi controllata usando come algoritmo di verifica pubblico la funzione ver_k. Lo scopo è ovviamente quello di certificare l'identità del firmatario, e questo deve poter essere fatto in tempi ragionevoli, pertanto si richiede che per ogni $k \in \mathcal{K}$ il computo delle funzioni sig_k e ver_k sia di tipo polinomiale.

Dopo aver già introdotto le firme digitali per il protocollo RSA nel Capitolo precedente, vediamo ora in dettaglio altri protocolli di firma digitale.

5.1 DSA

Il *Digital Signature Algorithm* (in breve DSA) pubblicato nel 1991 consiste in una efficace modifica di un precedente schema a firma digitale noto come *ElGamal Digital Signature* del 1985. La versione ufficialmente implementata prende il nome di DSS (*Digital Signature Standard*. Esiste anche uno schema basato sulle curve ellittiche che si chiama *Elliptic Curve Digital Signature Algorithm*, ECDSA, di cui non parleremo (il lettore interessato può però consultare [4]).

Alice, od una Trusted Autority TA, scegli due primi grandi p, q con la condizione che

$$q|p-1.$$

(Nelle implementazioni pratiche $2^{1000} < p < 2^{2000}$ e $2^{160} < q < 2^{320}$). Dopodiché sceglie un elemento $g \in \mathbb{F}_p^*$ d'ordine q (ad esempio se w è un generatore per $\mathbb{F}_p^*$, si può prendere $g = w^{\frac{p-1}{q}}$). Poi Alice sceglie una chiave $0 < \alpha < p-1$, calcola $h = g^\alpha$ e prende pubblici: p, q, g ed h. Il messaggio che Alice vuole firmare è un elemento $x \in \langle g \rangle \simeq \mathbb{Z}_q \subseteq \mathbb{F}_p^*$. Per firmare x sceglie un $0 < k < q$ e calcola gli elementi

$$S_1 = g^k \quad \text{ed} \quad S_2 \equiv (x + \alpha S_1)k^{-1} \pmod q$$

dove S_2 è il resto della divisione di $(x + \alpha S_1)k^{-1}$ per q. La funzione di firma è pertanto

$$\mathrm{sig}_{(\alpha,k)}(x) = (S_1, S_2) \in \mathbb{Z}_q \times \mathbb{Z}_q.$$

La verifica da parte di Bob che riceve $(x, (S_1, S_2))$ avviene in questo modo. Bob calcola separatamente le quantità

$$V_1 \equiv xS_1^{-1} \pmod q$$
$$V_2 \equiv S_1 S_2^{-1} \pmod q$$

quindi accetta la firma (S_2, S_2) sul messaggio x se e solo se

$$g^{V_1} h^{V_2} \equiv S_1 \pmod q.$$

Si noti che g ed h sono pubblici e V_1, V_2 sono stati calcolati da Bob. Se la firma è corretta lo schema funziona poiché

$$g^{V_1} h^{V_2} \equiv g^{xS_2^{-1}} g^{\alpha S_1 S_2^{-1}} \equiv g^{(x+\alpha S_1)S_2^{-1}} \equiv g^k$$

quindi

$$g^{V_1} h^{V_2} \equiv S_1 \pmod q.$$

Esercizio 5.1. Si provi che se Alice usa la stessa chiave (α, k) per firmare due messaggi x ed x', allora un attaccante Eva che intercetta $(x, (S_1, S_2))$ e $(x', (S_1', S_2'))$ riesce a risalire alla chiave (α, k).

5.2 Van Heyst and Pedersen Fail-stop Signature Scheme

Premettiamo la seguente Proposizione, la cui verifica lasciamo al lettore.

Proposizione 5.1. *Sia G un gruppo ciclico d'ordine primo p. Per ogni $\alpha = a + p\mathbb{Z} \in \mathbb{Z}_p$ ed ogni $x \in G$ la posizione*

$$x^\alpha := x^a$$

definisce un unico elemento di G. Inoltre, per ogni $\alpha, \beta \in \mathbb{Z}_p$ vale che $x^\alpha x^\beta = x^{\alpha+\beta}$ e, se $x \neq 1$, allora $x^\alpha = x^\beta$ se e solo se $\alpha = \beta$.

Lo schema di Van Heyst e Pedersen per le firme digitali, pubblicato nel 1992, prevede un'autorità super-partes che faccia da garante, quella che generalmente si chiama *Trusted Autority* (TA). Per prima cosa, la TA sceglie un primo grande p e un gruppo ciclico $G = \langle g \rangle$ d'ordine p in cui il DLP sia difficile da risolvere. Dopodiché sceglie un $\alpha \in \mathbb{Z}_p$, calcola $h = g^\alpha$ e pubblica (p, g, h), mentre tiene segreto α.

Il modello di firma prevede poi le seguenti scelte per gli spazi

$$\mathcal{P} = \mathbb{Z}_p, \quad \mathcal{F} = \mathbb{Z}_p^2, \quad \mathcal{K} \subseteq G^2 \times \mathbb{Z}_p^4,$$

dove ogni chiave $k \in \mathcal{K}$ è della forma $k = (x, y; \alpha_1, \alpha_2, \beta_1, \beta_2)$ con $x, y \in G, \alpha_i, \beta_i \in \mathbb{Z}_p$ e

$$\begin{cases} x = g^{\alpha_1} h^{\alpha_2} \\ y = g^{\beta_1} h^{\beta_2} \end{cases}$$

Le prime due componenti di k costituiscono la parte pubblica della chiave, mentre la parte restante è quella segreta. A questo punto la firma del messaggio m tramite la chiave $k = (x, y; \alpha_1, \alpha_2, \beta_1, \beta_2)$ è definita da

$$\mathrm{sig}_k(m) = (\sigma, \tau) \in \mathcal{F}$$

dove

$$\begin{cases} \sigma = \alpha_1 + m\beta_1 \\ \tau = \alpha_2 + m\beta_2 \end{cases}$$

Si noti che per firmare un messaggio serve la parte segreta della chiave.
La verifica di un messaggio firmato $(m, (\sigma, \tau))$ avviene in questo modo

$$\mathrm{ver}_k(m, (\sigma, \tau)) = \begin{cases} 1 & \text{se} \quad g^\sigma h^\tau = xy^m \\ 0 & \text{altrimenti} \end{cases}$$

Se infatti la firma è autentica abbiamo che

$$g^\sigma h^\tau = g^{\alpha_1 + m\beta_1} h^{\alpha_2 + m\beta_2} = g^{\alpha_1} h^{\alpha_2} g^{m\beta_1} h^{m\beta_2} = g^{\alpha_1} h^{\alpha_2} (g^{\beta_1} h^{\beta_2})^m = xy^m.$$

Quindi la funzione ver_k è ben definita.

Facciamo ora alcune osservazioni riguardo lo schema appena descritto.

1) Il procedimento di firma è monouso.

Supponiamo che con una stessa chiave $k = (x, y; \alpha_1, \alpha_2, \beta_1, \beta_2)$ vengano firmati due messaggi diversi, diciamo m ed $\overline{m}$. Allora sono trasmesse le firme

$$(\sigma, \tau) = (\alpha_1 + m\beta_1, \alpha_2 + m\beta_2) \quad \text{e} \quad (\overline{\sigma}, \overline{\tau}) = (\alpha_1 + \overline{m}\beta_1, \alpha_2 + \overline{m}\beta_2).$$

Un attaccante che intercetta i messaggi firmati ha a disposizione allora i seguenti dati: $m, \overline{m}, \sigma, \tau, \overline{\sigma}, \overline{\tau}$ ed il sistema

$$\begin{cases} \alpha_1 + m\beta_1 = \sigma \\ \alpha_2 + m\beta_2 = \tau \\ \alpha_1 + \overline{m}\beta_1 = \overline{\sigma} \\ \alpha_2 + \overline{m}\beta_2 = \overline{\tau} \end{cases}$$

nelle 4 incognite $\alpha_1, \alpha_2, \beta_1, \beta_2$. Poiché la matrice associata ha determinante $(m - \overline{m})^2 \neq 0$, l'attaccante riuscirebbe agevolmente a risolvere il sistema e quindi a conoscere la parte privata della chiave.

2) Cosa succede se un messaggio firmato $(m, (\sigma', \tau'))$ ha superato il test di verifica anche se la firma (σ', τ') è diversa dalla nostra (σ, τ).

Essendo il test di verifica stato superato abbiamo che

$$g^{\sigma'} h^{\tau'} = xy^m = g^{\sigma} h^{\tau},$$

da cui segue

$$g^{\sigma - \sigma'} = h^{\tau' - \tau}$$

e ricordando che $h = g^{\alpha}$,

$$g^{\sigma - \sigma'} = g^{\alpha(\tau' - \tau)}.$$

Ora per la Proposizione 5.1 si ha che

$$\sigma - \sigma' = \alpha(\tau' - \tau)$$

ovvero: se un falsario è riuscito a falsificare la nostra firma in modo da superare il test di verifica, allora noi possiamo ricavare α come

$$\alpha = \frac{\sigma - \sigma'}{\tau' - \tau}$$

cosa che, ricordiamo, per la scelta di p era molto difficile. A questo punto, abbiamo una prova concreta (cioè il valore di α) da presentare alla TA che la nostra firma è stata

falsificata.

3) Vediamo ora quale probabilità ha un falsario di trovare una chiave k' che firmi il messaggio m come nostra la chiave k.

Diremo che due chiavi sono equivalenti se hanno le prime due componenti (pubbliche) coincidenti. Le chiavi $\overline{k}$ equivalenti a $k = (x, y; \alpha_1, \alpha_2, \beta_1, \beta_2)$, con ovvie notazioni devono soddisfare al seguente sistema

$$\begin{cases} g^{\overline{\alpha_1}} g^{\alpha(\overline{\alpha_2})} = g^{\rho} \\ g^{\overline{\beta_1}} g^{\alpha(\overline{\beta_2})} = g^{\theta} \end{cases} \tag{5.1}$$

dove $\rho = \alpha_1 + \alpha\alpha_2$ e $\theta = \beta_1 + \alpha\beta_2$. Da (5.1) segue che

$$\begin{cases} \overline{\alpha_1} + \alpha\overline{\alpha_2} = \rho \\ \overline{\beta_1} + \alpha\overline{\beta_2} = \theta \end{cases} \tag{5.2}$$

la cui matrice associata ha rango 2. Ne segue che lo spazio delle soluzioni di (5.1) ha dimensione $4 - 2 = 2$, e poiché il campo è finito d'ordine p, possiamo concludere che ci sono esattamente p^2 chiavi $\overline{k}$ equivalenti a k. Un potenziale falsario, conoscendo la parte pubblica di k, incomincia a stilare un elenco delle p^2 chiavi equivalenti a k; da una di queste deve scegliere ora una chiave che firmi m come k. Non tutte queste però firmano il messaggio m come k, bensì solo quelle per cui vale che

$$\mathrm{sig}_{\overline{k}}(m) = \mathrm{sig}_k(m) = (\sigma, \tau).$$

Ne segue che se $\overline{k} = (x, y; \overline{\alpha_1}, \overline{\alpha_2}, \overline{\beta_1}, \overline{\beta_2})$ è equivalente a k e pone la stessa firma di k su m allora deve essere

$$\begin{cases} \overline{\alpha_1} + m\overline{\beta_1} = \sigma \\ \overline{\alpha_2} + m\overline{\beta_2} = \tau \\ \overline{\alpha_1} + \alpha\overline{\alpha_2} = \rho \\ \overline{\beta_1} + \alpha\overline{\beta_2} = \theta \end{cases} \tag{5.3}$$

La matrice associata al sistema (5.3) è

$$\begin{pmatrix} 1 & 0 & m & 0 \\ 0 & 1 & 0 & m \\ 1 & \alpha & 0 & 0 \\ 0 & 0 & 1 & \alpha \end{pmatrix}$$

che è facile vedere ha rango 3. Pertanto l'insieme delle soluzioni di (5.3) è una retta, e quindi consta di esattamente p elementi distinti. Nella sua scelta il falsario ha quindi una probabilità di successo pari a $\frac{p}{p^2} = \frac{1}{p}$, che è molto bassa essendo p un numero primo enorme.

5.3 Lo schema di Chaum - Van Antwerpen

Si tratta di uno schema di firma digitale incontrovertibile introdotto nel 1989 in [11]. La verifica della firma richiede la collaborazione del firmatario, qualora questi disconosca una sua firma, automaticamente lo schema prevede un protocollo di refutazione.

Al solito, sia p un primo grande e sia $G = \langle g \rangle$ un gruppo ciclico d'ordine p in cui il problema del logaritmo discreto sia difficile da risolvere. Il firmatario sceglie una chiave privata $\alpha \in \mathbb{F}_p$, $\alpha \neq 0, 1$ e rende esplicita la chiave pubblica (g, h), con $h = g^\alpha$. Un messaggio è un elemento $x \in G$ e la firma del messaggio è semplicemente

$$\mathrm{sig}_G(x) = x^\alpha.$$

Il messaggio firmato è pertanto la coppia

$$(x, s)$$

con $s = x^\alpha$.
La verifica della firma avviene in tre passi.

1. Il verificatore V, come si usa dire, lancia la sfida al firmatario F scegliendo una coppia $(u, v) \in \mathbb{F}_p^2$ e calcolando
$$z = s^u h^v.$$
Quindi spedisce z ad F.

2. Il firmatario F accoglie la sfida. Calcola $w = z^{\alpha^{-1}}$ e lo rispedisce a V.

3. V accetta la firma di F solo se
$$w = x^u g^v.$$
Si noti che per V questa è un'operazione fattibile in quanto x e g sono pubblici ed u, v sono stati scelti da V.

Vediamo nel dettaglio cosa succede algebricamente nel caso in cui la firma è autentica. Poiché

$$z = s^u h^v = (x^u g^v)^\alpha$$

si ha concretamente che

$$w = z^{\alpha^{-1}} = x^u g^v.$$

Esaminiamo ora quante chances può avere un falsario. Proviamo prima il seguente

Lemma 5.2. *Sia $G = \langle g \rangle$ un gruppo ciclico d'ordine primo p e sia $h = g^\alpha$, per qualche $\alpha \in \mathbb{F}_p$, $\alpha \neq 0, 1$. Scelti arbitrariamente due elementi $s, z \in G$, si ha che esistono esattamente p coppie $(u, v) \in \mathbb{F}_p^2$ per cui*

$$z = s^u h^v.$$

Inoltre se $x \in G$ ed $s \neq x^\alpha$, allora per ogni $w \in G$ esiste un'unica coppia $(\hat{u}, \hat{v}) \in \mathbb{F}_p^2$ tale che

$$\begin{cases} z = s^{\hat{u}} h^{\hat{v}} \\ w = x^{\hat{u}} g^{\hat{v}} \end{cases}$$

Dimostrazione. Posti $s = g^i$ e $z = g^j$, con $i, j \in \mathbb{F}_p$, la condizione $z = s^u h^v$ diventa

$$g^j = g^{iu+\alpha v},$$

ovvero

$$j = iu + \alpha v.$$

Questa rappresenta una retta nel piano $\mathbb{F}_p^2$, dunque contiene esattamente p punti.

Dimostriamo ora la seconda parte dell'enunciato. Usando le notazioni di sopra e scritti $x = g^l$ e $w = g^k$, con $l, k \in \mathbb{F}_p$, il sistema dell'enunciato è equivalente al seguente sistema di due equazioni in due incognite nel campo $\mathbb{F}_p$

$$\begin{cases} iu + \alpha v = j \\ lu + v = k \end{cases}$$

La matrice incompleta associata è dunque $\begin{pmatrix} i & \alpha \\ l & 1 \end{pmatrix}$, il cui determinate è $i - \alpha l$. Pertanto il sistema ammette un'unica soluzione se e solo se $i - \alpha l \neq 0$, ovvero se e solo se $s = g^i \neq g^{\alpha l} = x^\alpha$. ■

Ora un falsario, che non è in grado di risolvere un'istanza del logaritmo discreto, riceve la sfida z da V e deve produrre la risposta corretta w. Per il Lemma 5.2, il falsario sa che V ha scelto una delle possibili p coppie $(u, v) \in \mathbb{F}_p^2$ per cui $z = s^u h^v$. Per la seconda parte del Lemma però ognuna di queste dà luogo a un diverso w. Pertanto la cosa migliore che un falsario può fare è cercare di indovinare la risposta corretta fra p possibili scelte. La probabilità di successo è quindi $1/p$.

Disconoscimento. Chiediamoci ora se un firmatario può comportarsi in modo scorretto, ovvero se durante il processo di verifica ha la possibilità di disconoscere la propria firma dicendo che è falsa.
Supponiamo che durante la fase di verifica siano state generate la sfida $z = s^u h^v$ e la risposta w da parte del firmatario. Il procedimento di verifica però fallisce e il firmatario

afferma che la firma in questione è sbagliata. Vediamo in dettaglio come il verificatore può accertarsi della sua buona fede.

Si attua un secondo processo di verifica e viene lanciata una nuova sfida $z' = s^{u'}h^{v'}$. Il verificatore ottiene come risposta w' da parte del firmatario. Ora può controllare se

$$(wg^{-v})^{u'} = (w'g^{-v'})^{u}. \tag{5.4}$$

Se questa equazione vale, allora il verificatore accetta che il firmatario sta dicendo la verità e quindi che la firma iniziale è effettivamente falsa, in caso contrario egli sa, con probabilità almeno $1 - \frac{1}{p}$, che il firmatario sta mentendo. Proviamo quanto appena affermato. L'equazione (5.4) è sicuramente soddisfatta se il firmatario è onesto, poiché in tal caso avremmo che

$$(wg^{-v})^{u'} = (z^{\alpha^{-1}}g^{-v})^{u'} = ((s^{u}h^{v})^{\alpha^{-1}}g^{-v})^{u'} = (s^{u\alpha^{-1}}g^{v}g^{-v})^{u'} = s^{uu'\alpha^{-1}}$$
$$(w'g^{-v'})^{u} = (z'^{\alpha^{-1}}g^{-v'})^{u} = ((s^{u'}h^{v'})^{\alpha^{-1}}g^{-v'})^{u} = (s^{u'\alpha^{-1}}g^{v'}g^{-v'})^{u} = s^{uu'\alpha^{-1}}$$

Supponiamo invece che la firma sia corretta ma che il firmatario abbia voluto convincere il verificatore che fosse falsa. Allora avremo che $s = x^{\alpha}$, poiché la firma è valida, però alla prima verifica F decide di rispondere con un $w \neq z^{\alpha^{-1}} = x^{u}g^{v}$. Ciononostante, durante la seconda verifica F è costretto a presentare un w' che soddisfi la condizione (5.4). Il Lemma che segue mostra che il firmatario non ha idea quale w' scegliere, in quanto questi sono tutti equiprobabili.

Lemma 5.3. *Con le notazioni di sopra siano $s = x^{\alpha}$ e $w \neq x^{u}g^{v}$. Allora per ogni $w' \in G$ esiste esattamente un'unica coppia $(u', v') \in \mathbb{F}_p^2$ che soddisfa l'equazione (5.4).*

Dimostrazione. Supponiamo che un elemento w' di G soddisfi l'equazione (5.4) per una scelta $(u', v') \in \mathbb{F}_p^2$. Allora risolvendo l'equazione abbiamo che

$$w' = (x')^{u'}g^{v'}, \tag{5.5}$$

dove $x' = (wg^{-v})^{u^{-1}}$. Ma la (5.5) è esattamente l'equazione di verifica per la firma s sul messaggio x', per la scelta di (u', v') da parte di V. Ora s non può essere una firma valida per x', poiché lo è per x (ovvero se fosse $s = (x')^{\alpha}$ avremmo che $x^{\alpha} = (x')^{\alpha}$, da cui segue $x = x'$, e pertanto $x = (wg^{-v})^{u^{-1}}$, e $x^{u}g^{v} = w$, contrariamente all'ipotesi). Pertanto $s \neq (x')^{\alpha}$ e per il Lemma 5.2 per ogni $w_1 \in G$ esiste un'unica coppia $(u_1, v_1) \in \mathbb{F}_p^2$ tale che

$$\begin{cases} z' = s^{u_1}h^{v_1} \\ w_1 = (x')^{u_1}g^{v_1} \end{cases}$$

In particolare in corrispondenza di $w_1 = w'$, l'ultima condizione è verificata se e solo se $w' = (wg^{-v})^{u_1u^{-1}}g^{v_1}$ e questo accade se e solo se la condizione (5.4) è soddisfatta con

(u_1, v_1) al posto di (u', v'). Ne segue che, una volta scelto $w' \in G$, la (5.4) è soddisfatta per una sola coppia possibile. ■

Il firmatario, durante il protocollo di disconoscimento, è pertanto costretto a presentare al verificatore l'unico elemento $w' \in G$ che rende valida la (5.4). Egli pertanto ha una sola scelta possibile fra le p possibili. In conclusione se il primo round ha dato risposta negativa è ragionevole che il verificatore si fidi del protocollo di refutazione e giudichi falsa la firma, qualora l'uguaglianza (5.4) sia soddisfatta.

5.4 Le funzioni Hash

Sia Σ un alfabeto arbitrario. Se l è un intero naturale, indichiamo con Σ^l l'insieme i cui elementi sono le stringhe di lunghezza al più l sull'alfabeto Σ e poniamo inoltre

$$\Sigma^* = \bigcup_{l \in \mathbb{N}} \Sigma^l.$$

Definizione 18. *Siano $m, l \in \mathbb{N}$, con $m > l$.*

- *Una funzione $h : \Sigma^m \longrightarrow \Sigma^l$ si dice* funzione di compressione.
- *Una funzione $h : \Sigma^* \longrightarrow \Sigma^l$ si dice* funzione hash.

In particolare, una funzione hash prende stringhe di lunghezza arbitraria e le comprime in qualcosa di lunghezza finita. Non sono mai funzioni iniettive.

Le funzioni hash, così come le funzioni di compressione, sono molto usate in diversi contesti. In Crittografia ad esempio vengono molto impiegate nelle firme digitali. Pensiamo di voler autentificare un documento x. Spesso questo può trattarsi di una stringa estremamente lunga; conviene pertanto pubblicare una funzione hash h e firmare $h(x)$ invece che x, così il documento firmato sarà $\big(x, \mathrm{sig}(h(x))\big)$.

Ovviamente non tutte le funzioni hash, o di compressione, sono adatte a svolgere con sicurezza il ruolo che loro compete; ci sono delle richieste fondamentali. La prima per una funzione h di questo tipo è di praticità, ovvero che h sia facilmente calcolabile. In secondo luogo, si richiede che la funzione h sia di tipo *One-Way* (in breve OW).

Una *collisione* per la funzione h è una coppia (x, x') per cui $x \neq x'$ e $h(x) = h(x')$.

Definizione 19. *Una funzione hash h si dice* debolmente senza collisioni *(DSC) se, dato x nel dominio di h, è difficile trovare x' tale che la coppia (x, x') è una collisione per h. Diremo invece che h è* fortemente senza collisioni *(FSC) se è difficile trovare una collisione per h.*

Proposizione 5.4. *Sia h una funzione hash o di compressione. Se h è fortemente senza collisioni, allora h è one-way.*

DIMOSTRAZIONE. Limitiamoci al caso h sia una funzione di compressione,

$$h : \Sigma^m \longrightarrow \Sigma^l,$$

con $m > l$ e supponiamo che h non sia OW. Possiamo pertanto supporre che esista un oracolo Θ per l'inversione locale di h. Vogliamo provare che h non è FSC e lo faremo costruendo un algoritmo di tipo probabilistico che, con alta probabilità, permette di trovare collisioni. Procediamo con il seguente schema.

1. Scegli uniformemente un $x \in \Sigma^m$ e calcola $h(x)$.
2. Interroga l'oracolo Θ, ottenendo x' tale che $h(x') = h(x)$.
3. Se $x' \neq x$ termina, altrimenti vai al punto 1.

Mostriamo ora che la probabilità di successo di questo algoritmo è maggiore della metà. Ripetendo l'algoritmo un numero k di volte la probabilità di insuccesso sarà minore di $\frac{1}{2^k}$, e quindi, basta prendere k sufficientemente grande per trovare delle collisioni.

Indichiamo con $\sim$ la relazione di equivalenza associata ad h, ovvero $a \sim b$ se e solo se $h(a) = h(b)$; siano inoltre $\mathcal{C}$ l'insieme quoziente $\Sigma^m/\sim$ e per ogni x in Σ^m, indichiamo con $[x]$ la classe di equivalenza contenente x ovvero $[x] = \{a \in \Sigma^m | h(a) = h(x)\}$. Ora ogni volta che viene scelto un elemento x, l'oracolo Θ risponde sempre con un $x' \in [x]$, e la probabilità di successo è molto alta essendo $\frac{|[x]|-1}{|[x]|}$. In generale la probabilità di successo dell'intero algoritmo è data da

$$\begin{aligned}\sum_{C\in\mathcal{C}} \frac{|C|-1}{|\Sigma^m|} &= \frac{1}{|\Sigma|^m} \sum_{C\in\mathcal{C}} (|C|-1) = \frac{1}{|\Sigma|^m}\Big(\sum_{C\in\mathcal{C}} |C| - \sum_{C\in\mathcal{C}} 1\Big) \\ &\geq \frac{1}{|\Sigma|^m}\big(|\Sigma|^m - |\Sigma|^l\big) = 1 - \frac{1}{|\Sigma|^{m-l}} \geq \frac{1}{2},\end{aligned}$$

non appena $m > l$ poiché $|\Sigma| \geq 2$. ■

5.4.1 Hash di Chaum, van Heyst, Pfitzman

Sia q un numero primo di Sophie-Germain, ovvero un primo tale che anche $2q+1$ è un numero primo. Indichiamo con $p = 2q+1$ e con G il gruppo ciclico moltiplicativo $\mathbb{F}_p^*$, sia inoltre α un generatore per G. Un elemento appartenente all'intervallo $[0, q^2)$ ha una scrittura in binario di circa $2\log_2 q$ cifre, invece un elemento di $[0, p)$ in generale sarà lungo $\log_2 p \approx \log_2 q$ bits, cioè circa la metà. Costruiamo una funzione di compressione

$h : [0, q^2) \longrightarrow [1, p)$. Scriviamo gli elementi di $[0, q^2)$ in forma q-adica, ovvero per il generico $x \in [0, q^2)$ poniamo $x = x_1 + x_2 q$ con $x_1, x_2 \in [0, q)$. Fissato un elemento $\beta = \alpha^a \in G$, per un'opportuna scelta di $a \in [1, p)$, poniamo:

$$\begin{aligned} h : [0, q^2) &\longrightarrow [1, p) = G \\ x = x_1 + x_2 q &\longmapsto \alpha^{x_1} \beta^{x_2}. \end{aligned}$$

La funzione h ovviamente non può essere iniettiva, però è fortemente senza collisioni, infatti trovare collisioni per h è equivalente a risolvere un'istanza del problema del logaritmo discreto. Proviamo nel dettaglio quanto appena enunciato.

Supponiamo d'avere un oracolo in grado di determinare le collisioni per h e mostriamo che siamo in grado di risolvere il DLP in tempo polinomiale.
Sia $(x, y) = (x_1 + x_2 q, y_1 + y_2 q)$ una collisione per h. Allora, poiché $\beta = \alpha^a$,

$$\begin{aligned} \alpha^{x_1} \beta^{x_2} &= \alpha^{y_1} \beta^{y_2} \\ \alpha^{x_1 - y_1} &= \alpha^{a(y_2 - x_2)} \end{aligned}$$

che è vera se e solo se

$$x_1 - y_1 \equiv a(y_2 - x_2) \pmod{p-1}. \tag{5.6}$$

Sia d il massimo comune divisore $d = (y_2 - x_2, p - 1)$. Se $d = 1$ allora $y_2 - x_2$ è un elemento invertibile di $\frac{\mathbb{Z}}{(p-1)\mathbb{Z}}$, pertanto preso $c \in \mathbb{Z}$ tale che $c(y_2 - x_2) \equiv 1 \pmod{p-1}$, abbiamo che

$$a \equiv c(x_1 - y_1) \pmod{p-1}.$$

Se invece $d \neq 1$, allora d è un divisore non-banale di $p - 1 = 2q$, ed essendo x_2 ed y_2 minori di q, necessariamente si ha che $d = 2$. La (5.6) equivale allora a

$$\frac{x_1 - y_1}{2} \equiv a \frac{y_2 - x_2}{2} \pmod{q}. \tag{5.7}$$

Poiché ora $(\frac{y_2 - x_2}{2}, q) = 1$, tramite l'algoritmo di Euclide si può determinare un intero e tale che $e \cdot \frac{y_2 - x_2}{2} \equiv 1 \pmod{q}$. Da (5.7) otteniamo

$$a \equiv e \cdot \frac{x_1 - y_1}{2} \pmod{q}.$$

Detto r il resto della divisione di $e \cdot \frac{x_1 - y_1}{2}$ per q, possiamo scrivere

$$a = r + tq,$$

con $t \in \mathbb{N}$. Essendo $a \in [0, p)$, si ha che $t \in \{0, 1\}$. Si provano entrambi i valori e si guarda quello che combacia. Come si vede il procedimento sopra esposto è di tipo polinomiale.

Viceversa, supponiamo ora di avere un oracolo per il logaritmo discreto e mostriamo come in tempo polinomiale si riescono a trovare collisioni per h. Tramite l'oracolo troviamo $a = \log_\alpha \beta$. Preso un arbitrario $x \in [0, q^2)$ e scritto $x = x_1 + x_2 q$, detto $z = h(x)$, abbiamo che

$$z = h(x) = \alpha^{x_1} \beta^{x_2} = \alpha^k,$$

dove $k = x_1 + a x_2$. Abbiamo pertanto che (x, x') è una collisione per h se e solo se scritto $x' = x'_1 + x'_2 q$, le coordinate (x'_1, x'_2) sono soluzioni dell'equazione diofantea

$$X_1 + a X_2 = k \tag{5.8}$$

in $\mathbb{F}_p$. Poiché queste soluzioni si determinano in tempo polinomiale abbiamo la tesi.

5.4.2 Un potenziale attacco alle funzioni Hash

In questa sezione presentiamo una procedura semplice ed efficace per attaccare le funzioni Hash, che sfrutta il cosiddetto *"paradosso dei compleanni"*: la probabilità che almeno due persone in un gruppo compiano gli anni lo stesso giorno è largamente superiore a quanto potrebbe dire l'intuito; già in un gruppo di 23 persone la probabilità è circa il 50%.

In particolare, proveremo che se una funzione Hash viene costruita in maniera ingenua, l'algoritmo in questione è in grado sempre di trovare collisioni con probabilità superiore al 50%.

Sia

$$h : X \longrightarrow Y$$

una finzione Hash e sia $|Y| = m$. Dato un numero $r \in [0, 1]$, vogliamo trovare il minimo intero $k \in \mathbb{N}$ per cui assegnati arbitrariamente k elementi distinti di X, $x_1, x_2, \ldots, x_k$, ci sia una collisione con probabilità r, ovvero con probabilità r esistono $i \neq j$ tali che $h(x_i) = h(x_j)$.

Indichiamo con P la probabilità di scegliere k elementi di Y tutti distinti. Questa è data da

$$P = \frac{m-1}{m} \cdot \frac{m-2}{m} \cdot \ldots \cdot \frac{m-k+1}{m} = \prod_{i=1}^{k-1} \left(1 - \frac{i}{m}\right).$$

Sfruttando la maggiorazione $1 + x \leq \mathrm{e}^x$, per ogni $x \in \mathbb{R}$, abbiamo che

$$P \leq \prod_{i=1}^{k-1} \mathrm{e}^{-i/m} = \mathrm{e}^{\sum_{i=1}^{m-1} -i/m} = \mathrm{e}^{-\frac{k(k-1)}{2m}}.$$

Cerchiamo ora k per cui

$$\mathrm{e}^{-\frac{k(k-1)}{2m}} < 1 - r. \tag{5.9}$$

La (5.9) è soddisfatta se e solo se

$$k^2 - k + 2m \log(1 - r) > 0 \tag{5.10}$$

la cui soluzione per k positivo è

$$k > \frac{1}{2}\left(1 + \sqrt{1 - 8m \log(1 - r)}\right) \approx c\sqrt{m},$$

con c costante reale positiva. Si noti che già per $r = 1/2$, si ha che $c \approx 1$, quindi bastano circa $\sqrt{m}$ valori Hash per garantire l'esistenza di una collisione con probabilità del 50%. Ne segue che affinché la funzione Hash h sia FSC, necessariamente $m = |Y|$ deve essere scelto sufficientemente grande.

Capitolo 6

Protocolli di sicurezza

6.1 Condivisione dei segreti

Supponiamo che un gruppo di persone abbia la necessità di custodire un segreto in modo tale che per potervi accedere sia richiesta una soglia minima di consensi da parte del gruppo. Una tipica situazione è quella di un gruppo di n dipendenti di un'azienda, ciascuno dei quali possiede una propria chiave di autenticazione, con la quale poter accedere alla propria postazione di lavoro. L'azienda fornisce le chiavette di autenticazione e se un dipendente perde la propria chiave, per poter generare i codici per una chiavetta sostitutiva, serve necessariamente il consenso da parte di tutti gli altri $n-1$ membri del gruppo. Una macchina speciale è in grado di generare la chiave sostitutiva solo dopo aver preso visione di tutte le $n-1$ chiavi restanti.

6.1.1 Lo schema a soglia di Shamir

Sia n il numero di utenti coinvolti nello schema e sia k la soglia minima per la condivisione del segreto. Sia $\mathbb{F}$ un campo finito di cardinalità maggiore di n e sufficientemente grande da rendere intrattabile una ricerca esaustiva. Il segreto da condividere è rappresentato da un elemento $a \in \mathbb{F}$. L'autorità che gestisce il segreto sceglie un polinomio $f \in \mathbb{F}[x]$ di grado $k-1$ e tale che $f(0) = a$:

$$f = a_{k-1}x^{k-1} + \ldots + a_1 x + a_0,$$

(con $a_0 = a$). Poi sceglie n elementi distinti di $\mathbb{F}$, siano questi $\alpha_i \in \mathbb{F}$, per $i = 1, 2, \ldots, n$, e consegna ad membro i-esimo del gruppo la coppia $(\alpha_i, f(\alpha_i))$. Ora ogni k-upla $(i_1, \ldots, i_k)$ di persone è in grado di risolvere il sistema lineare

$$\left\{ \sum_{l=0}^{k-1} a_l (\alpha_{i_j})^l = f(\alpha_{i_j}) \qquad \text{per } j = 1, \ldots, k \right. \tag{6.1}$$

di k equazioni nelle k incognite a_l. Infatti tale sistema si scrive in termini matriciali come

$$\begin{pmatrix} 1 & \alpha_{i_1} & (\alpha_{i_1})^2 & \dots & (\alpha_{i_1})^{k-1} \\ 1 & \alpha_{i_2} & (\alpha_{i_2})^2 & \dots & (\alpha_{i_2})^{k-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha_{i_k} & (\alpha_{i_k})^2 & \dots & (\alpha_{i_k})^{k-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_k \end{pmatrix} = \begin{pmatrix} f(\alpha_{i_1}) \\ f(\alpha_{i_2}) \\ \vdots \\ f(\alpha_{i_k}) \end{pmatrix}$$

Si noti che la matrice del sistema è una matrice di Vandermonde, il cui determinante è pertanto

$$\prod_{s<r} (\alpha_{i_r} - \alpha_{i_s}) \neq 0$$

essendo gli α_{i_j} tutti distinti. Invertendo la matrice si trovano tutti i coefficienti del polinomio e quindi anche il termine noto del polinomio, cioè il segreto. Osserviamo che le informazioni in possesso di un numero $< k$ persone non bastano a risolvere il sistema. Infatti, anche nel caso di $k-1$ persone, avremmo da risolvere un sistema lineare di $k-1$ equazioni in k incognite. L'insieme delle soluzioni è pertanto una retta affine isomorfa al campo $\mathbb{F}$, quindi è come se fossimo nella condizione di partenza, ci sono tante scelte possibili per il segreto quanti sono gli elementi del campo.

Una variante dello schema di Shamir fa uso del cosiddetto polinomio interpolante di Lagrange. La formula di interpolazione di Lagrange è una formula esplicita per (l'unico) polinomio $L_f(x)$ di grado $\leq k$ che assume il valore $f(\alpha_{i_j})$ nel punto α_{i_j}, per $j = 1, 2, \dots, k$, α_{i_j} tutti distinti. Nel nostro caso in cui f stesso è un polinomio, è ovvio che $L_f = f$, in generale le formula per il polinomio di Lagrange è data da

$$L_f(x) = \sum_{j=1}^{k} f(\alpha_{i_j}) L_{i_j}(x) \quad \text{dove} \quad L_{i_j}(x) = \prod_{1 \leq t \leq k,\, t \neq j} \frac{x - \alpha_{i_t}}{\alpha_{i_j} - \alpha_{i_t}}. \tag{6.2}$$

Gli L_{i_j} sono detti pesi. Si noti che $L_{i_j}(\alpha_{i_t}) = \delta_{j,t}$ e quindi $L_f(\alpha_{i_j}) = f(\alpha_{i_j})$ per ogni $j = 1, \dots, k$ e come detto sopra, per il principio di identità dei polinomi, $L_f(x) = f(x)$. L'uso di L_f rende però i conti più agevoli. Infatti poiché siamo interessati solo al termine noto di f, possiamo sostituire $x = 0$ nella formula (6.2) e otteniamo che

$$a = \sum_{j=1}^{k} b_j f(\alpha_{i_j}),$$

con

$$b_j = \prod_{1 \leq t \leq k,\, t \neq j} \frac{\alpha_{i_t}}{\alpha_{i_t} - \alpha_{i_j}} \quad \text{per } j = 1, 2, \dots, k.$$

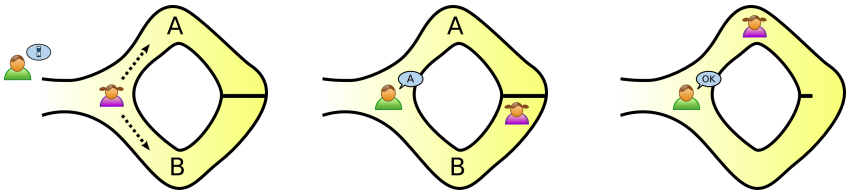
Si noti che i b_j possono essere calcolati a priori e i loro valori non sono necessariamente segreti.

Molto spesso è utile dare un peso ai diversi membri del gruppo. Ad esempio se si tratta di membri di una società è logico supporre che chi dispone di un numero maggiore di azioni debba avere un potere maggiore. Per questo motivo esistono efficaci varianti dello schema a soglia sopra rappresentato.

6.2 Protocolli a conoscenza zero

Un protocollo a conoscenza zero è una procedura che permette ad un soggetto di dimostrare ad un altro soggetto la veridicità di una certa affermazione, senza per forza rivelare l'affermazione stessa.

L'esempio tipico di protocollo a conoscenza zero è la cosiddetta *caverna di Alì Babà* (si veda [5]). Alice vuole dimostrare a Bob di essere in grado di aprire la porta della caverna di Alì Babà senza rivelare la parola segreta. La caverna ha la seguente struttura: dopo un ingresso la caverna di suddivide in due rami, che si riuniscono in un punto diametralmente opposto rispetto all'entrata.



Nel punto in cui i due rami si riuniscono c'è una porta che si apre con la parola segreta. Alice e Bob si accordano sul seguente metodo. Alice entra nella caverna senza che Bob possa vedere da che lato è entrata. A questo punto Bob si affaccia all'ingresso della caverna e chiede ad Alice di uscire da uno dei due lati urlando 'destra' o 'sinistra'. Se Alice esce dal lato richiesto Bob accetta il protocollo, altrimenti lo rifiuta. Dato che ogni esecuzione del protocollo è indipendente, se il protocollo viene ripetuto k volte la probabilità che Alice riesca ad imbrogliare Bob tutte le volte è esattamente $\frac{1}{2^k}$. Come si vede, dal protocollo Bob può pienamente convincersi che Alice sia effettivamente in possesso della conoscenza richiesta; lui però non sarà mai in grado di apprendere il segreto che fa aprire la porta.

6.2.1 Il protocollo di Feidge-Fiat-Shamir

Sia n il prodotto di due numeri primi distinti, $n = pq$. Il problema di estrarre le radici quadrate nell'anello $\mathbb{Z}_n$ è computazionalmente equivalente a quello di fattorizzare n (si veda ad esempio [8], 8.4.5), pertanto possiamo metterci nelle ipotesi che questo sia un problema intrattabile con i mezzi a disposizione. Alice e Bob scelgono un certo $y \in U(\mathbb{Z}_n)$, ed Alice sostiene di conoscere la radice quadrata di y, ovvero di conoscere un elemento $s \in \mathbb{Z}_n$ tale che $s^2 = y$. Alice vuole convincere Bob della sua conoscenza senza però rivelare a Bob s. Si mettono d'accordo sul seguente protocollo.

1. Alice sceglie un $r_1 \in U(\mathbb{Z}_n)$ e calcola $r_2 = r_1^{-1}s$. Poi svolge $r_1^2 =: x_1$ e $r_2^2 =: x_2$, e invia (x_1, x_2) a Bob.

2. Bob verifica che $x_1x_2 = y$. Quindi sceglie $t \in \{x_1, x_2\}$ e chiede ad Alice di fornirgli una radice di t.

3. Alice risponde con
$$z := \begin{cases} r_1 & \text{se } t = x_1 \\ r_2 & \text{se } t = x_2. \end{cases}$$

4. Bob controlla che $z^2 = t$.

Se Alice conosce effettivamente s non sorgono problemi e il test funziona sempre. Se invece Alice mente e non conosce s, allora ha una probabilità del 50% di rispondere esattamente al test. Infatti Alice sceglie r_1 e, non conoscendo s, invia a Bob $x_1 = (r_1)^2$ e $x_2 = yx_1^{-1}$. Bob controlla che $x_1x_2 = y$ e chiede ad Alice di fornirgli una radice per $t \in \{x_1, x_2\}$. Ora se $t = x_1$, Alice non ha problemi nel rispondere con r_1, se invece $t = x_2$ Alice, non conoscendo $r_2 = r_1^{-1}s$, è costretta a trovare un elemento z tale che $z^2 = x_2$, e tale operazione è intrattabile nel gruppo $\mathbb{Z}_n$, ne segue che in questo caso Alice non sa che rispondere. Pertanto solo nel 50% dei casi (cioè per la scelta di $t = x_1$) Alice sa dare una risposta a Bob. Ovviamente il test viene ripetuto un numero sufficiente, k, di volte, e la probabilità di avere sempre successo per Alice scende drasticamente a $\frac{1}{2^k}$. È ragionevole supporre quindi che se Alice risponde sempre esattamente a tutti i test, allora conosce veramente la radice di y. Si noti che in tutti i passaggi, le informazioni che arrivano a Bob non permettono mai di risalire ad s, a meno che Bob non sia in grado di estrarre radici quadrate in $U(\mathbb{Z}_n)$.

Abbiamo visto che per essere convincente questo test necessita di essere ripetuto più volte. Vediamo ora una variazione del test che permette di diminuire il numero di iterazioni.

Alice sceglie $s_1, s_2, \ldots, s_k \in U(\mathbb{Z}_n)$ e calcola e rende pubblici gli elementi $v_i = s_i^{-2}$, per $i = 1, 2, \ldots, k$. Tramite il seguente protocollo Alice può dimostrare di conoscere tutti gli s_i.

1. Alice sceglie un arbitrario $r \in U(\mathbb{Z}_n)$, calcola $x = r^2$ che spedisce a Bob.
2. Bob sceglie un vettore $\underline{b} = (b_1, b_2, \ldots, b_k) \in \{0,1\}^k$ che invia ad Alice.
3. Alice calcola l'elemento
$$y = r \prod_{i=1}^{k} s_i^{b_i},$$
che rispedisce a Bob.
4. Bob verifica se
$$y^2 \prod_{i=1}^{k} v_i^{b_i} = x.$$

In caso di risposta esatta il protocollo termina con successo per Alice.
Ovviamente nel caso in cui Alice conosca effettivamente la sequenza $(s_i)_i$, al momento della verifica abbiamo che
$$y^2 \prod_{i=1}^{k} v_i^{b_i} = r^2 \prod_{i=1}^{k} s_i^{2b_i} \prod_{i=1}^{k} v_i^{b_i} = r^2 \prod_{i=1}^{k} (s_i^2 v_i)^{b_i} = r^2 = x.$$
Se invece Alice sta mentendo e non conosce tutti gli s_i, allora Alice potrebbe scegliere in partenza un $y \in U(\mathbb{Z}_n)$ e una sequenza di bit $\underline{c} = (c_1, c_2, \ldots, c_k)$, calcolare $x = y^2 \prod_{i=1}^{k} v_i^{c_i}$ ed inviare x a Bob. Questi sceglie poi la sua sequenza di bit $(b_1, b_2, \ldots, b_k)$ e, dopo che ha ricevuto y da Alice, verifica se
$$y^2 \prod_{i=1}^{k} v_i^{b_i} = x = y^2 \prod_{i=1}^{k} v_i^{c_i}.$$
Ma questa è soddisfatta se e solo se
$$\prod_{i=1}^{k} v_i^{b_i - c_i} = 1, \tag{6.3}$$
e tale equazione può essere vera anche se gli esponenti $b_i - c_i$ non sono tutti nulli. Infatti $U(\mathbb{Z}_n)$ è un gruppo abeliano finito, pertanto si scrive, in modo unico a meno dell'ordine, come prodotto diretto di gruppi ciclici finiti. Posto
$$U(\mathbb{Z}_n) \simeq C_1 \times C_2 \times \ldots \times C_m$$
e detto α_i un generatore per C_i, $i = 1, 2, \ldots, m$, abbiamo che se scegliamo i v_i della forma $(1, \ldots, 1, \alpha_i, 1, \ldots, 1)$, siamo sicuri che la (6.3) è soddisfatta se e solo se $b_i = c_i$ per ogni $i = 1, 2, \ldots, k$. Utilizzando questa accortezza la probabilità che Alice scelga esattamente la stringa di bit giusta è $\frac{1}{2^k}$.

Capitolo 7

Fattorizzazione di interi

7.1 Test di primalità

Un test di primalità consente di determinare se un dato numero intero è primo oppure no, senza necessariamente esibire una sua fattorizzazione.

Un metodo banale per vedere se un intero n è primo consiste nel provare a dividere n per tutti gli interi compresi tra 2 e $\sqrt{n}$. Ci si accorge però che questa ricerca esaustiva diventa presto intrattabile al crescere di n, da qui la necessità di trovare metodi alternativi efficaci e più pratici. Per tutto il XX secolo non erano noti test di primalità che girassero in tempo polinomiale, se non test di tipo probabilistico. Nel 2003 M. Agrawal, N. Kayal e N. Saxena in [1] hanno trovato un test di primalità polinomiale e deterministico, dimostrando che il problema di capire se un intero è un primo non è così difficile come si riteneva che fosse. L'analisi del loro algoritmo è fuori dagli scopi del corso. Presenteremo invece alcuni metodi alternativi alla ricerca esaustiva, mostrandone i limiti, e poi ci concentreremo sull'algoritmo di Solovay-Strassen, che è di tipo polinomiale probabilistico.

7.1.1 Eventuali metodi basati sul Teorema di Wilson e sul "piccolo Teorema di Fermat"

Una nota proprietà dei numeri primi è fornita dal seguente teorema di Wilson.

Teorema 7.1. *Un numero naturale n è primo se e solo se $(n-1)! \equiv 1$ (mod n).*

Dimostrazione. Si vedano le dispense del corso di Algebra 1 ([9], Teorema 8.11). ■ Il Teorema presenta una proprietà che caratterizza completamente i numeri primi.

L'unico inconveniente però consiste proprio nel verificare questa proprietà, in quanto calcolare $(n-1)!$ diventa presto intrattabile al crescere di n. Un altro eventuale criterio

si basa sull'altrettanto celebre

Teorema 1.1 [piccolo Teorema di Fermat] *Se n è un numero primo allora per ogni intero a coprimo con n vale che $a^{n-1} \equiv 1$ (mod n).*

Il Teorema di Fermat potrebbe essere utilizzato come test di tipo probabilistico. Si prende n e si incominciano ad elevare alla potenza $(n-1)$-esima i vari interi minori di n. Se per un certo a l'enunciato del Teorema non vale, allora certamente possiamo dedurre che n non è primo. Se però ogni tentativo va a buon fine, non siamo in grado di dire niente sulla primalità di n. Infatti, per un noto teorema di W. R. Alford, A. Granville e C. Pomerance ([2]), sappiamo che esistono infiniti numeri non primi n per cui vale che $a^{n-1} \equiv 1$ (mod n), per ogni a coprimo con n. Sono i cosiddetti numeri di Carmichael di cui abbiamo ampiamente parlato nel Capitolo 1, sezione 1.3.

7.1.2 Il test di Solovay-Strassen

Si tratta di un test di tipo probabilistico che gira in tempo polinomiale, e prova che o un numero è composto, oppure è molto probabilmente un numero primo. Si basa sul concetto di pseudoprimo di Eulero e sul fatto che controllare che un numero è pseudoprimo di Eulero rispetto a una qualche base è un'operazione computazionalmente fattibile. Richiamiamo i concetti già enunciati nel Capitolo 1.

Definizione 20. *Sia n un intero naturale dispari e composto e sia $a \in \mathbb{Z}$, coprimo con n. Diremo che n è* pseudoprimo di Eulero rispetto alla base *a se*

$$\left(\frac{a}{n}\right) \equiv a^{\frac{n-1}{2}} \quad (mod\ n).$$

Ricordiamo inoltre questo importante risultato, già dimostrato nel Capitolo 1.

Teorema 1.21 *Sia n un intero naturale composto. Allora esiste almeno un intero a coprimo con n, rispetto al quale n non è pseudoprimo di Eulero.*

Notazione. Se $\alpha = a + n\mathbb{Z} \in \mathbb{Z}_n$ denotiamo con

$$\left(\frac{\alpha}{n}\right) := \left(\frac{a}{n}\right) + n\mathbb{Z}.$$

Possiamo allora affermare che n è pseudoprimo di Eulero rispetto alla base $\alpha \in U(\mathbb{Z}_n)$ se e solo se $\alpha^{\frac{n-1}{2}} = \left(\frac{\alpha}{n}\right)$.

Proposizione 7.2. *Se n non è pseudoprimo di Eulero rispetto ad una certa base, allora esistono almeno metà elementi di $U(\mathbb{Z}_n)$ rispetto ai quali n non è pseudoprimo di Eulero.*

Dimostrazione. Sia n un numero dispari e composto. Per il Teorema 1.21, esiste almeno una base per cui n non è pseudoprimo di Eulero. Definiamo

$$\mathcal{A} = \left\{ \alpha \in U(\mathbb{Z}_n) \middle| \alpha^{\frac{n-1}{2}} = \left(\frac{\alpha}{n}\right) \right\} \quad \text{e} \quad \mathcal{B} = U(\mathbb{Z}_n) \setminus \mathcal{A}$$

e mostriamo che $|\mathcal{B}| \geq \frac{1}{2}|U(\mathbb{Z}_n)| = \frac{1}{2}\varphi(n)$. Per ipotesi $\mathcal{B}$ è non vuoto, sia $\beta \in \mathcal{B}$ e poniamo

$$\begin{aligned} f : \mathcal{A} &\longrightarrow \mathcal{B} \\ \alpha &\longmapsto \alpha\beta \end{aligned}$$

f è una funzione ben definita. Infatti se per assurdo esistesse un $\alpha \in \mathcal{A}$ per cui $f(\alpha) \notin \mathcal{B}$ allora si avrebbe

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

e poiché $\alpha^{\frac{n-1}{2}} = \left(\frac{\alpha}{n}\right)$, avremmo che $\beta^{\frac{n-1}{2}} = \left(\frac{\beta}{n}\right)$, ovvero $\beta \in \mathcal{A}$, che è assurdo.
È altresì immediato verificare che f è iniettiva. Pertanto $|\mathcal{B}| \geq |\mathcal{A}| = |U(\mathbb{Z}_n) \setminus \mathcal{B}|$, da cui segue che $|\mathcal{B}| \geq \frac{1}{2}\varphi(n)$. ■

Abbiamo ora tutti gli strumenti per vedere nel dettaglio come funziona il test di Solovay-Strassen.
Sia dato un intero dispari n. Si sceglie un parametro k, che sarà il numero di iterazioni dell'algoritmo.

1. Si sceglie un $a \in (0, n)$.
2. Si calcola $d = (a, n)$. Se $d \neq 1$ abbiamo trovato un fattore non banale di n e l'algoritmo finisce. Altrimenti $d = 1$ e si passa al punto 3.
3. Si vede se è soddisfatta $a^{\frac{n-1}{2}} = \left(\frac{\alpha}{n}\right)$. Se non lo è, allora n è composto e l'algoritmo termina. Altrimenti si torna al punto 1.

Se supponiamo che l'algoritmo esegua k iterazioni senza mai arrestarsi, abbiamo che la probabilità che n sia composto è piuttosto bassa. Infatti definiamo i seguenti eventi:

$C :=$ un intero dispari n arbitrario ma "di grandezza specificata" è composto.

$D :=$ dopo k iterazioni l'algoritmo risponde "n è primo".

In virtù della Proposizione 7.2, sappiamo che la probabilità condizionata $P(D|C) \leq \frac{1}{2^k}$. Noi però siamo interessati a $P(C|D)$, questa si può calcolare tramite la formula di Bayes

$$P(C|D) = \frac{P(D|C) \cdot P(C)}{P(D)} = \frac{P(D|C) \cdot P(C)}{P(D|C)P(C) + P(D|\overline{C})P(\overline{C})}$$

dove $\overline{C}$ indica l'evento complementare a C. Supponiamo ora che n sia di grandezza specificata, ovvero che sia $N \leq n \leq 2N$. Ricordiamo il celebre

Teorema dei numeri primi. *Sia x un intero positivo. Allora il numero $\pi(x)$ dei primi compresi tra 1 ed x è asintotico a $\frac{x}{\ln(x)}$.*

Per una dimostrazione si veda ad esempio [3].
Stimiamo ora il numero dei primi (dispari) compresi tra N e $2N$. Per il Teorema appena enunciato, questo si approssima con

$$\frac{2N}{\ln(2N)} - \frac{N}{\ln(N)} \approx \frac{N}{\ln(N)} \approx \frac{n}{\ln(n)}.$$

Poiché ci sono $\frac{N}{2} \approx \frac{n}{2}$ numeri dispari tra N e $2N$, ne segue che

$$P(\overline{C}) \approx \frac{n/\ln(n)}{n/2} = \frac{2}{\ln(n)} \quad \text{e} \quad P(C) \approx 1 - \frac{2}{\ln(n)}.$$

Pertanto, tenuto conto che $P(D|\overline{C}) = 1$, abbiamo che

$$\begin{aligned} P(C|D) &\approx \frac{P(D|C) \cdot \left(1 - \frac{2}{\ln(n)}\right)}{P(D|C) \cdot \left(1 - \frac{2}{\ln(n)}\right) + \frac{2}{\ln(n)}} = \frac{P(D|C) \cdot (\ln(n) - 2)}{P(D|C) \cdot (\ln(n) - 2) + 2} \\ &\leq \frac{2^{-k}(\ln(n) - 2)}{2^{-k}(\ln(n) - 2) + 2} = \frac{\ln(n) - 2}{\ln(n) - 2 + 2^{k+1}}. \end{aligned}$$

Ne segue che $P(C|D)$ tende a zero abbastanza velocemente. Nella pratica per essere abbastanza sicuri basta prendere $k = 50$ o $k = 100$.

Analizziamo ora in termini di prestazioni l'algoritmo di Solovay-Strassen.
Ad ogni iterazione dobbiamo calcolare un M.C.D., questo richiede $O(\text{size}\,(n)^3)$ o.e. Poi dobbiamo effettuare delle potenze modulari; anche queste richiedono al massimo $O(\text{size}\,(n)^3)$ o.e. Per il calcolo del simbolo di Jacobi invece osserviamo che ciò non si può fare direttamente dalla definizione (in quanto non conosciamo la fattorizzazione di n) dobbiamo pertanto utilizzare ripetutamente il Teorema di reciprocità quadratica per i simboli di Jacobi (Teorema 1.20 e la discussione che ne segue). Anche in questo caso il costo computazione è $O(\text{size}\,(n)^3)$. Pertanto il costo di ogni iterazione è $O(\text{size}\,(n)^3)$.

7.2 Fattorizzazione di interi

Come già osservato nel Capitolo 4, al momento non sono noti algoritmi di fattorizzazione che girano in tempo polinomiale su computer attuali. Presenteremo di seguito tre algoritmi diversi di fattorizzazione.

7.3 Il metodo ρ di Pollard

Questo algoritmo, pubblicato nel 1975, viene anche chiamato *"metodo di Monte Carlo"* ed è particolarmente efficace quando vogliamo fattorizzare un intero positivo che possiede un fattore primo relativamente piccolo.

Sia n l'intero naturale che vogliamo fattorizzare. Si sceglie una funzione

$$f : \mathbb{Z}_n \longrightarrow \mathbb{Z}_n$$

che sia facile da calcolare, generalmente un polinomio di grado ≥ 2 (molto spesso x^2+1) ed un "punto di partenza" $x_0 \in \mathbb{Z}_n$. Si costruisce poi la successione

$$x_{i+1} = f(x_i), \qquad \text{per } i = 0, 1, 2, \ldots$$

Dopodiché si confrontano i diversi x_i con la speranza di trovare due classi distinte modulo n che però coincidono modulo un divisore proprio di n. L'idea è quella di stimare la frequenza con cui la successione $\{x_i\}_i$ tende a formare un ciclo e quindi poi di confrontarla con quella dell'analoga sequenza, $\{x_i \pmod p\}_i$, modulo un qualche eventuale $p|\sqrt{n}$. Per quest'ultima sequenza sarà più probabile trovare una ripetizione, una volta trovata allora la successione formerà un ciclo[1]. Affinché l'algoritmo possa funzionare è necessario che la sequenza $\{x_i \pmod p\}_i$ si ripeta prima della sequenza $\{x_i\}_i$. Il problema consiste pertanto nel trovare due indici, $j < k$ per cui $x_j \neq x_k$ e $(x_k - x_j, n) \neq 1$ (dove al solito si identificano gli interi $< n$ con gli elementi di $\mathbb{Z}_n$). Si noti che il fatto che f sia di tipo polinomiale modulo n garantisce che se $a \equiv b \pmod p$ allora $f(a) \equiv f(b) \pmod p$ per ogni eventuale divisore p di n. Vediamo un esempio per chiarire il metodo.

Esempio 6. Sia $n = 91$ e scegliamo $f(x) = x^2 + 1$ e $x_0 = 1$. Allora $x_1 = f(x_0) = f(1) = 2$, $x_2 = f(x_1) = f(2) = 5$ e $x_3 = f(x_2) = f(5) = 26$. Pertanto $x_3 \neq x_2$ e $(x_3 - x_2, n) = (21, 91) = 7 \neq 1$, trovando la fattorizzazione $91 = 7 \cdot 13$.

La probabilità con cui una successione di questo tipo si ripete è messa in luce dal seguente

Lemma 7.3. *Sia S un insieme finito di r elementi e sia $S^S = \{f | f : S \longrightarrow S\}$. Siano inoltre $\lambda \in \mathbb{R}_+$ ed $l = 1 + [\sqrt{2\lambda r}]$. Detti*

$$U = \big\{(f, x_0) | f \in S^S,\, x_0 \in S\big\} = S^S \times S$$
$$V = \{(f, x_0) \in U | x_0, x_1, \ldots x_l \textit{ sono tutti distinti}\}$$

(dove $x_{i+1} = f(x_i)$), vale che

$$p = \frac{|V|}{|U|} < e^{-\lambda}.$$

[1] Il nome "ρ" dell'algoritmo deriva proprio dalla forma della lettera greca che ricorda il grafo orientato formato dai valori della successione che si chiudono su un ciclo.

Dimostrazione. Abbiamo che $|U| = \left|S^S\right| \cdot |S| = r^r \cdot r = r^{r+1}$ mentre

$$|V| = r \cdot (r-1) \cdot \ldots \cdot (r-l) \cdot r^{r-l} = r^{r-l} \cdot \prod_{i=0}^{l}(r-i)$$

poiché abbiamo esattamente r scelte per x_0, $(r-1)$ per x_1, ... ,$(r-l)$ per x_l, mentre r possibili scelte per completare l'arbitraria funzione appartenente a V. Ne segue che

$$p = \frac{r^{r-l} \cdot \prod_{i=0}^{l}(r-i)}{r^{r+1}} = \prod_{i=0}^{l}\left(1 - \frac{i}{r}\right).$$

Sfruttando la diseguaglianza $\ln(1-x) < -x$, per ogni $x \in (0,1)$, otteniamo che

$$\ln(p) = \sum_{i=0}^{l} \ln\left(1 - \frac{i}{r}\right) < \sum_{i=0}^{l}\left(-\frac{i}{r}\right) = -\frac{l(l-1)}{2r} < -\frac{(l-1)^2}{2r} < -\frac{(\sqrt{2\lambda r})^2}{2r} = -\lambda$$

e quindi $p < \mathrm{e}^{-\lambda}$. ■

Come dicevamo il Lemma fornisce una stima circa il probabile tempo di attesa per una ripetizione nella sequenza $\{x_i\}_i$, a patto che la funzione f sia un polinomio che si comporti "in modo casuale" (si veda l'esercizio 7.1).

L'algoritmo passa ora a calcolare, per ogni scelta di k, il massimo comune divisore $(x_k - x_j, n)$, per ogni $j < k$. Al crescere di k, il numero di operazioni da compiere diventa presto improponibile. Vediamo come la scelta polinomiale per f è in grado di ridurlo sensibilmente.

Supponiamo che esistano $j_0 < k_0$ tali che $x_{j_0} \neq x_{k_0}$ e $(x_{j_0} - x_{k_0}, n) \neq 1$ e mostriamo che se j e k sono tali che $k - j = k_0 - j_0$ allora $(x_k - x_j, n) \neq 1$.
Detti $m = k - k_0 = j - j_0$ e $d = (x_{j_0} - x_{k_0}, n) \neq 1$, applicando f^m a $x_{k_0} \equiv x_{j_0} \pmod{d}$ abbiamo che

$$f^m(x_{k_0}) \equiv f^m(x_{j_0}) \pmod{d}$$

ovvero

$$f^{k-k_0}(f^{k_0}(x_0)) \equiv f^{j-j_0}(f^{j_0}(x_0)) \pmod{d}$$

da cui segue

$$x_k \equiv f^k(x_0) \equiv f^j(x_0) \equiv x_j \pmod{d}.$$

Possiamo pertanto concludere elencando la procedura in questione.
Si parte con $k = 1$ e si calcola x_k.
Se $2^h \leq k < 2^{h+1}$ si pone $j = 2^h - 1$ e si calcola $d = (x_k - x_j, n)$.
Se $d \neq 1$, allora d è un fattore di n e l'algoritmo termina.
Altrimenti si ripete il procedimento con $k + 1$ al posto di k.

Esercizio 7.1. Sia S un insieme non vuoto di cardinalità r e denotiamo con U_1 l'insieme $U_1 = \{(f, x_0) | f \in Sym(S),\ x_0 \in S\}$, dove $Sym(S)$ indica la collezione di tutte le biezioni di S in sé. Detti $x_{i+1} = f(x_i)$, per ogni $(f, x_0) \in U_1$ sia k il più piccolo indice per cui esiste $j < k$ tale che $f(x_k) = f(x_j)$. Si provi quanto segue.

1. k puó essere r e per ogni $t \in \{1, 2, \ldots, r\}$ la probabilità $P(k = t) = \frac{1}{r}$.
2. Il valor medio di k è $\frac{r+1}{2}$.
3. Se f è un polinomio di primo grado il metodo "ρ" di Pollard è poco funzionale.

Nota storica. Dal punto di vista storico, forse il risultato più significativo finora raggiunto con il metodo "ρ" consiste nella fattorizzazione dell'ottavo numero di Fermat $F_8 = 2^{2^8} + 1$ ad opera di R. P. Brent e J. M. Pollard nel 1980 ([7]).

$$\begin{aligned} F_8 = {} & 1238926361552897 * \\ & 93461639715357977769163558199606896584051237541638188580280321. \end{aligned}$$

7.4 Il metodo di Fermat

Esiste un modo metodo efficiente per fattorizzare un intero dispari che è il prodotto di due interi vicini tra di loro ed è basato sulla seguente

Proposizione 7.4. *Sia n un intero dispari. Allora esiste una corrispondenza biunivoca tra le fattorizzazioni di n come $n = ab$, con $a \geq b > 0$ e le rappresentazioni di n come $n = t^2 - s^2$, con $t, s \geq 0$.*

Dimostrazione. Si considerino le mappe $(a, b) \longmapsto (\frac{a+b}{2}, \frac{a-b}{2})$ e $(t, s) \longmapsto (t+s, t-s)$. $\square$

Se n si fattorizza come $n = ab$ con a e b numeri (ignoti ma) tra loro vicini, allora $s = (a - b)/2$ è piccolo, dunque t è poco più grande di $\lfloor \sqrt{n} \rfloor$. Si può allora pensare di trovare la fattorizzazione di n provando tutti i valori di t partendo da $t = \lfloor \sqrt{n} \rfloor + 1$, $\lfloor \sqrt{n} \rfloor + 2$,... finché $t^2 - n$ non risulta un quadrato perfetto, in tal caso avremo che $t^2 - n = s^2$ e dunque $(t + s, n) \neq 1$ sarebbe un fattore per n (N.B. Stiamo assumendo che n non sia un quadrato perfetto).

Esempio 7. Si fattorizzi $n = 200819$.
Partiamo da $t = \lfloor \sqrt{200819} \rfloor + 1 = 449$. Allora

$$t^2 - n = (449)^2 - 200819 = 782 \notin \mathbb{N}^2.$$

Quindi prendiamo $t = 450$ e notiamo che

$$t^2 - n = (450)^2 - 200819 = 1681 = (41)^2 \in \mathbb{N}^2,$$

da cui segue la fattorizzazione

$$200819 = (450 + 41)(450 - 41) = 491 \cdot 409.$$

Si noti che $491 \simeq 409$.

Nel caso invece in cui a e b sono distanti, il metodo di Fermat è parecchio lungo. Si può però generalizzare nel seguente modo.
Scegliamo un intero piccolo k e consideriamo $t = \lfloor\sqrt{kn}\rfloor + 1$, $t = \lfloor\sqrt{kn}\rfloor + 2$, ... finché $t^2 - kn$ sia un quadrato perfetto. Allora avremmo che, scritto $t^2 - kn = s^2$, da $(t + s)(t - s) = kn$, il massimo comun divisore $(t + s, n)$ fornirebbe un fattore di n.

Esercizio 7.2. Fattorizzare $n = 141467$. (Suggerimento: prendere $k = 3$).

Vediamo ora come generalizzando queste idee di Fermat si riesce ad ottenere un algoritmo di fattorizzazione abbastanza soddisfacente.

Osserviamo che se n è l'intero dispari che vogliamo fattorizzare, se troviamo due interi b e c tali che

$$b^2 \equiv c^2 \pmod{n}$$

con $b \not\equiv c \pmod{n}$, allora automaticamente un fattore proprio di n si determina calcolando $(b + c, n)$ (o equivalentemente $(b - c, n)$). L'idea di fondo per determinare dei candidati per b e c, consiste nello scegliere diversi numeri $\{b_i\}_{i\in I}$ con la proprietà che ogni $b_i^2 \pmod{n}$ è un prodotto di primi relativamente piccoli (diciamo minori di una certa soglia) e per i quali esiste un sottoinsieme $J \subseteq I$ tali che $\prod_{j\in J} b_j$ è un quadrato perfetto modulo n. Vediamo ora i dettagli.

Ricordiamo che se n è un intero naturale per ogni $a \in \mathbb{Z}$ il *residuo assoluto di a modulo n* è l'unico intero $r \in (-n/2, +n/2]$ tale che $a \equiv r \pmod{n}$. Denoteremo tale numero semplicemente con $a \bmod n$.

Definizione 21. *Una* base di fattori *(o* factor bases*) è un insieme finito non vuoto della forma $B = \{p_1, p_2, \ldots, p_h\}$, dove $p_1 \in \mathbb{P} \cup \{-1\}$ e $p_i \in \mathbb{P}$ sono tutti distinti.*
*Dato un intero n ed una base di fattori B, diremo che un intero a è un B-*numero per *n se $a^2 \bmod n$ può essere scritto come prodotto di numeri appartenenti a B.*

Esempio 8. Siano $n = 4633$ e $B = \{-1, 2, 3\}$. Allora i numeri $67, 68$ e 69 sono tutti B-numeri per n. Infatti $67^2 \equiv -144 = -1 \cdot 2^4 \cdot 3^2 \bmod n$, $68^2 \equiv -9 = -1 \cdot 3^2 \bmod n$, $69^2 \equiv 128 = 2^7 \bmod n$.

Dato n intero che vogliamo fattorizzare, scegliamo B una base di fattori di cardinalità diciamo h e definiamo la seguente mappa

$$\begin{aligned} f : \{B\text{-numeri}\} &\longrightarrow \mathbb{F}_2^h \\ b &\longmapsto (\epsilon_1, \epsilon_2, \ldots, \epsilon_h) \end{aligned}$$

dove, una volta scritto $b^2 \bmod n = \prod_{i=1}^{h} p_i^{\alpha_i}$, per ogni $i = 1, \ldots, h$, si ha che $\epsilon_i = \alpha_i \pmod 2$.

Esempio 9. Riprendendo l'esempio 8 abbiamo che $f(67) = (1, 0, 0)$, $f(68) = (1, 0, 0)$ e $f(69) = (0, 1, 0)$.

Supponiamo ora di trovare un sottoinsieme finito $\{b_j\}_{j \in J}$ dell'insieme di tutti i B-numeri per cui valga

$$\sum_{j \in J} f(b_j) = \underline{0}. \tag{7.1}$$

Indichiamo per ogni $j \in J$ con a_j il residuo assoluto di b_j^2 modulo n e scriviamolo come B-numero, ovvero:

$$a_j = \prod_{i=1}^{h} p_i^{\alpha_{ji}}.$$

Abbiamo allora che posto $f(b_j) = (\epsilon_{j1}, \epsilon_{j2}, \ldots, \epsilon_{jh})$, per ogni $j \in J$, vale $\epsilon_{ji} = \alpha_{ji} \pmod 2$, per $i = 1, \ldots, h$. La condizione (7.1) equivale allora a

$$\sum_{j \in J} (\epsilon_{j1}, \epsilon_{j2}, \ldots, \epsilon_{jh}) = \underline{0} \in \mathbb{F}_2^h.$$

ovvero a

$$\sum_{j \in J} \epsilon_{ji} = 0 \in \mathbb{F}_2 \quad \text{per ogni } i = 1, \ldots, h,$$

cioè $\sum_{j \in J} \alpha_{ji}$ è pari per ogni $i = 1, \ldots, h$. Sia pertanto $\sum_{j \in J} \alpha_{ji} = 2\gamma_i$ per ogni $i = 1, \ldots, h$, ne segue che

$$\prod_{j \in J} a_j = \prod_{j \in J} \prod_{i=1}^{h} p_i^{\alpha_{ji}} = \prod_{i=1}^{h} p_i^{\sum_{j \in J} \alpha_{ji}} = \prod_{i=1}^{h} p_i^{2\gamma_i}. \tag{7.2}$$

Possiamo allora definire i seguenti prodotti di residui assoluti mod n

$$b = \prod_{j \in J} (b_j \bmod n) \quad \text{e} \quad c = \prod_{i=1}^{h} (p_i^{\gamma_i} \bmod n)$$

La relazione (7.2), ed il fatto che gli a_j sono per definizione il residuo assoluto di b_j^2 modulo n, implicano che

$$b^2 \equiv c^2 \pmod n.$$

Ora se accade che $b \not\equiv \pm c \pmod n$ allora possiamo concludere trovando un fattore proprio non banale di n semplicemente calcolando $(b + c, n)$. Altrimenti se $b \equiv c \pmod n$ siamo sfortunati e dobbiamo in prima istanza cercare un nuovo insieme J_1 di b-numeri che svolga il ruolo di J, eventualmente allargare la nostra soglia B.

Esempio 10. Continuando con il nostro esempio si noti che

$$f(67) + f(68) = (1, 0, 0) + (1, 0, 0) = (0, 0, 0)$$

Pertanto calcoliamo $b = 67 \cdot 68 \bmod 4633 = -77$ e $c = (-1)^{\gamma_1} 2^{\gamma_2} 3^{\gamma_3} = -2^2 3^2 = -36$. Fortunatamente abbiamo che $-77 \not\equiv \pm 36 \pmod{4633}$ e quindi troviamo che $(-77 + 36, 4633) = 41$ è un fattore di 4633.

Alcune osservazioni sono d'obbligo.

Osservazione 6. Se per ogni $j \in J$ abbiamo che $b_j \leq \sqrt{\frac{n}{2}}$ allora $a_j = b_j^2$ e pertanto $f(b_j) = \underline{0}$ per ogni $j \in J$. In questo caso il procedimento sopra descritto porta ad una congruenza banale, i.e. $b = c$.

Osservazione 7. Poiché $\mathbb{F}_2^h$ è uno spazio vettoriale di dimensione h sul campo di due elementi $\mathbb{F}_2$, ogni volta che prendiamo $h+1$ vettori questi sono linearmente indipendenti. Quindi alla peggio dobbiamo generare $h+1$ differenti B-numeri per ottenere un insieme del tipo $\{b_j\}_{j \in J}$ che soddisfi la condizione (7.1).

Osservazione 8. Osserviamo infine che, modulo l'accortezza di scegliere i b_j abbastanza grandi (vedi l'osservazione di sopra), la probabilità che sia $b \equiv \pm c \pmod{n}$ è generalmente abbastanza piccola, almeno $\leq 1/2$.
Supponendo che n si fattorizzi come $n = \prod_{l=1}^{r} q_l^{\beta_l}$ con $r \geq 2$, ogni quadrato, b^2, di $\mathbb{Z}_n$ ha 2^r radici. Questo può essere dimostrato applicando il Teorema cinese del resto (Teorema 1.3):

$$\mathbb{Z}_n \simeq \mathbb{Z}_{q_1^{\beta_1}} \times \mathbb{Z}_{q_2^{\beta_2}} \times \ldots \times \mathbb{Z}_{q_r^{\beta_r}}.$$

Poichè si assume che b^2 è un elemento invertibile di $\mathbb{Z}_n$ (altrimenti avremmo già un fattore proprio di n, semplicemente facendo (b, n)), ogni $b^2 \pmod{q_i^{\beta_i}}$ è invertibile. Per il Lemma 1.10, ogni $U(\mathbb{Z}_{q_i^{\beta_i}})$ è un gruppo ciclico, in particolare quindi vi sono esattamente due radici distinte di $b^2 \pmod{q_i^{\beta_i}}$. Se ne conclude che le radici di b^2 in $\mathbb{Z}_n$ sono 2^r. La probabilità che sia $b \equiv \pm c \pmod{n}$ è allora $2^{-r+1} \leq 1/2$.

Si può provare che il costo computazionale dell'algoritmo è $O\left(\mathrm{e}^{C\sqrt{\log_2(n)\log_2(\log_2(n))}}\right)$ (il lettore interessato può consultare [18], V.3).

7.5 Il metodo $p - 1$ di Pollard

Questo metodo è particolarmente adatto quando il numero n che vogliamo fattorizzare ha un divisore primo p tale che $p - 1$ ha fattori primi piccoli, diciamo $p - 1 = \prod_i q_i^{a_i}$ dove per ogni i vale $q_i^{a_i} \leq B$ per qualche soglia B. Ovviamente a priori non sappiamo se n ha un primo di questo e pertanto bisogna procedere a tentativi. Vediamo come funziona l'algoritmo.

1. Scegli una soglia $B \in \mathbb{N}$ ed un intero k che sia multiplo di tutti gli interi $\leq B$ (ad esempio $k = B!$ oppure meglio $k = m.c.m.\{m|m \leq B\}$).
2. Scegli un intero $a \in \{2, 3, \ldots, n-2\}$ (ad es. $a = 2$ o $a = 3$).
3. Calcola $a^k \pmod{n}$ (ad esempio col metodo del Fast Powering Algorithm).
4. Calcola $d = (a^k - 1, n)$ (usando Euclide).
5. Se $d \neq n$ termina, altrimenti torna a 2. o ad 1.

Vediamo perché funziona. Essendo k un multiplo di tutti gli interi $\leq B$, k è multiplo di tutte le potenze di primi che fattorizzano $p-1$ e quindi $p-1$ divide k. Per il piccolo Teorema di Fermat (Teorema 1.1) abbiamo che $a^k \equiv 1 \pmod{p}$. Ne segue che

$$p|(a^k - 1, n) = d$$

e quindi $d \neq 1$ e dobbiamo solo preoccuparci che d non sia n.

Esempio 11. Fattorizziamo $n = 540143$ col metodo $p-1$ di Pollard. Scegliamo $B = 8$ come soglia. Sia $k = m.c.m.\{1, 2, 3, 4, 5, 6, 7, 8\} = 840$ e prendiamo $a = 2$. Abbiamo che

$$a^{840} \pmod{540143} \equiv 53047$$

e $(53046, 540143) = 421$ fornisce la fattorizzazione

$$540143 = 421 \cdot 1283.$$

Si noti che $p - 1 = 420 = 2^2 \cdot 3 \cdot 5 \cdot 7$.

Il grosso punto debole dell'algoritmo è che ogni speranza di successo viene posta nel gruppo $\mathbb{Z}_p^*$ al variare dei primi p che dividono n. Se per ogni divisore primo p di n il gruppo $\mathbb{Z}_p^*$ è divisibile da primi grandi (o potenze grandi di primi) questo metodo di fattorizzazione diventa impraticabile. Esiste una variante dovuta a Lenstra del metodo $p-1$ che si applica molto bene ai gruppi abeliani che sono associati a curve ellittiche su campi finiti (si veda il Capitolo **??**, ...).

Bibliografia

[1] M. Agrawal, N. Kayal, N. Saxena, *PRIMES is in P*, Ann. of Math. (2) **160** (2004), no. 2, 781–793.

[2] W. R. Alford, A. Granville, C. Pomerance, *There are Infinitely Many Carmichael Numbers*, Annals of Mathematics **139** (1994) 703–722.

[3] T. M. Apostol, *Introduction to Analytic Number Theory*, Springer-Verlag, New York, 1976.

[4] ANSI-ECDSA, *Pubblic key cryptography for the financial services industry: The elliptic curve digital signature algorithm (ECDSA)*, ANSI Report X9.62, American National Standards Institute, 1998.

[5] T. A. Berson, L. C. Guillou, J.-J. Quisquater, *How to Explain Zero-Knowledge Protocols to Your Children*, Advances in Cryptology – CRYPTO, **435** (1989) 628–631.

[6] D. Boneh, G. Durfee, *Cryptanalysis of RSA with private key d less than* N^{292}. IEEE Trans. Inform. Theory **46** (2000), no. 4, 1339–1349.

[7] R. P. Brent, J. M. Pollard, *Factorization of the eighth Fermat number*, Math. Comp. **36** (1981), no. 154, 627–630.

[8] J. Buchmann, *Introduction to cryptography. Second edition.* Undergraduate Texts in Mathematics. Springer-Verlag, New York, 2004.

[9] C. Casolo, *Dispense del corso di Algebra 1*, `web.math.unifi.it/users/casolo/dispense/algebra1_14.pdf`

[10] C. Casolo, *Dispense del corso di Algebra 2*, `http://web.math.unifi.it/users/casolo/dispense/algebra2_13.pdf`

[11] D. Chaum, H. van Antwerpen, *Undeniable Signatures*, LNCS **435** (1990), 212–216.

[12] J. Chernick, *On Fermat's simple theorem*, Bull. Amer. Math. Soc. **45**, (1939) 269–274.

[13] P. Cintioli, C. Toffalori, *Logica matematica*, 2000, McGraw-Hill.

[14] H. Davenport, *Aritmetica superiore*, Zanichelli, Bologna, 1994, ISBN 88-08-09154-6.

[15] Federal Information Processing Standards Publication 81, *DES modes of operation*, `http://csrc.nist.gov/publications/fips/fips81/fips81.htm`

[16] G. H. Hardy, *Ramanujan*, (1940) New York: Cambridge University Press.

[17] A. A. Karatsuba, *The Complexity of Computations*, Proceedings of the Steklov Institute of Mathematics **211**: 169–183. Translation from Trudy Mat. Inst. Steklova, **211**, 186–202 (1995).

[18] N. Koblitz, *A course in number theory and cryptography. Second edition.* Graduate Texts in Mathematics, **114**. Springer-Verlag, New York, 1994.

[19] D. E. Knuth, *The art of computer programming. v.2.*, Addison-Wesley Publ. Co.

[20] IEEE Communications Magazine Homepage, IEEE Communications Society, Comsoc.org. (Retrieved on 2013-10-29).

[21] F. Lemmermeyer, *Reciprocity Laws: From Euler to Eisenstein*, (2000) Berlin: Springer.

[22] D. Mundici, *Dalla macchina di Turing a P/NP*, 2013, McGraw-Hill.

[23] C. D. Olds, *Frazioni Continue*, Zanichelli, 1992.

[24] D. Shanks, *Class number, a theory of factorization and genera*, Proc. Symp. Pure Math. **20**, (1971), 415–440.

[25] P. W. Shor, *Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer*, SIAM J. Comput. **26** (5) 1484–1509.

[26] A. M. Turing, *On Computable Numbers, with an Application to the Entscheidungs problem*, Proceedings of the London Mathematical Society. **42**, 2 (1937) 230–265.

[27] Wikipedia,
`https://www.wikipedia.org`