POLITECNICO DI TORINO

Corso di Laurea Magistrale
in Ingegneria Matematica

Tesi di Laurea Magistrale

Codici correttori e loro applicazione alla crittografia post-quantum

Relatori
prof. Danilo Bazzanella
dott. Nadir Murru
firma dei relatori

. . . . . .
. . . . . .

Candidato
Mario Tamietti

firma del candidato

. . . . . .

Anno Accademico 2018-2019

Sommario

Il sistema crittografico oggi più diffuso è l'RSA. Tale sistema si basa sul fatto che, dato il prodotto di due numeri interi, è molto difficile risalire ai fattori di partenza. Questa sicurezza è però minata dal (probabile) avvento imminente del computer quantistico, su cui è possibile eseguire un algoritmo computazionalmente efficiente (l'algoritmo di Shor) per fattorizzare un intero. Per ovviare a questa minaccia, è stato introdotto un nuovo ramo di ricerca, detto crittografia post-quantum, che studia quegli algoritmi in grado di funzionare su computer classici ma resistenti anche ad attacchi quantistici. Fa parte di questo ramo la cosiddetta code-based cryptography, cioè l'insieme di quegli algoritmi crittografici che si basano sui codici correttori. Il primo è stato il sistema di McEliece, introdotto nel 1978 ma a cui all'inizio non era stata data grande considerazione. Esistono molteplici codici su cui basarsi per costruire un sistema crittografico; l'idea che sviluppiamo in questa tesi è quella di usare il codice di Fibonacci, studiato per la prima volta da Stakhov nel 1999. Il percorso seguito inizia con la presentazione di RSA e la sua spiegazione dal punto di vista dell'aritmetica modulare. Si prosegue poi con la dimostrazione di come il computer quantistico, usando l'algoritmo di Shor, sia in grado di romperlo. Vengono, a questo punto introdotti i codici correttori, di cui viene data una breve descrizione, soffermandosi in particolare sul codice di Goppa, che è quello usato nel sistema di McEliece. Nell'ultimo capitolo viene presentato un altro codice, quello di Fibonacci, per poi concludere esaminando i problemi che ostacolano la costruzione di un sistema crittografico basato su tale codice.

Indice

I Sistema RSA e computer quantistico 1

1 Crittografia a chiave pubblica e RSA 3
- 1.1 Introduzione: una definizione formale di Crittografia . . . 3
- 1.2 Aritmetica modulare . . . 4
  - 1.2.1 Quoziente e resto della divisione tra interi . . . 4
  - 1.2.2 Congruenza in modulo . . . 5
  - 1.2.3 Addizione, sottrazione e moltiplicazione modulari . . . 5
  - 1.2.4 Inverso modulare moltiplicativo . . . 6
  - 1.2.5 Algoritmo di Euclide . . . 6
  - 1.2.6 Algoritmo di Euclide esteso . . . 7
  - 1.2.7 Calcolo dell'inverso . . . 9
  - 1.2.8 Elevamento modulare a potenza . . . 9
  - 1.2.9 Teorema cinese del resto . . . 10
  - 1.2.10 Funzione di Eulero . . . 11
  - 1.2.11 Gruppo, anello e campo finito . . . 12
- 1.3 Crittografia a chiave privata . . . 13
  - 1.3.1 Problemi della crittografia a chiave privata . . . 14
- 1.4 Crittografia a chiave pubblica . . . 15
- 1.5 Sistema crittografico RSA . . . 16
  - 1.5.1 Funzionamento di RSA nel dettaglio . . . 17
  - 1.5.2 Sicurezza . . . 19
  - 1.5.3 Attacchi ad un sistema RSA . . . 20

2 Algoritmo di Shor 23
- 2.1 Introduzione . . . 23
- 2.2 Quantum Computing . . . 23
  - 2.2.1 Sistema quantistico a n stati . . . 24
- 2.3 Algoritmo di Shor . . . 25
  - 2.3.1 La funzione dell'algoritmo di Shor . . . 25
  - 2.3.2 Passaggi dell'algoritmo . . . 26
- 2.4 Effetti su RSA . . . 28
  - 2.4.1 Storia del computer quantistico . . . 29

II Crittografia post-quantum: sistema di McEliece e codice di Fibonacci **31**

3 Sistema Crittografico di McEliece 33
3.1 Introduzione . . . 33
3.2 Campi finiti . . . 33
3.2.1 Costruzione di campi finiti . . . 36
3.2.2 Moltiplicazione in un campo finito . . . 37
3.3 Codici correttori . . . 38
3.3.1 Distanza e peso di un codice . . . 41
3.3.2 Matrice di controllo parità e matrice generatrice . . . 42
3.4 Codice di Goppa . . . 44
3.4.1 Parametri del codice di Goppa . . . 45
3.4.2 Matrice di controllo parità e matrice generatrice . . . 47
3.4.3 Codifica e decodifica di un messaggio . . . 48
3.4.4 Correzione di errori . . . 49
3.5 Sistema crittografico di McElice . . . 53
3.5.1 Attacchi al sistema di McEliece . . . 55
3.5.2 Vantaggi e svantaggi . . . 57

4 Codice di Fibonacci 59
4.1 Introduzione . . . 59
4.2 Matrice Q di Fibonacci . . . 61
4.3 p numeri di Fibonacci . . . 63
4.4 Matrici di Fibonacci generalizzate . . . 64
4.4.1 Elevamento a potenza delle matrici di Fibonacci . . . 65
4.5 Codice di Fibonacci . . . 66
4.5.1 Relazioni tra gli elementi della matrice C . . . 67
4.5.2 Riconoscimento e correzione di errori . . . 68
4.5.3 Confronto con altri codici . . . 70
4.6 Implementazione di un sistema crittografico di Fibonacci . . . 71
4.6.1 Foma chiusa e limite degli Fp . . . 71
4.6.2 Correzione di errori (idea) . . . 73

Parte I

Sistema RSA e computer quantistico

Capitolo 1

Crittografia a chiave pubblica e RSA

1.1 Introduzione: una definizione formale di Crittografia

Con il termine *crittografia* si intende il processo di cifratura e decifratura di messaggi in un codice segreto, in modo da renderli incomprensibili a tutti, eccetto che al legittimo destinatario. Si distingue dalla *crittoanalisi*, che è invece l'arte di rompere un codice cifrato. La scienza che studia la comunicazione sicura e segreta, che comprende sia crittografia sia crittoanalisi, è detta *crittologia*.
I principi crittologici sono applicati alle reti televisive, informatiche e non solo; rivestono una grande importanza anche per le comunicazioni di tipo bancario, governativo e commerciale britannica.
Un sistema crittografico è basato su due operazioni Scala [2019]:

- **cifratura**, cioè il passaggio da un testo in chiaro P (plaintext) ad un testo cifrato C (cyphertext) per mezzo di un algoritmo di cifratura E (encrypter) e di una chiave K;
- **decifratura**, ossia il passaggio inverso, da un testo cifrato C a un testo in chiaro P per mezzo di un algoritmo di decifratura D (decrypter) e di una chiave K, eventualmente anche diversa da quella usata precedentemente.

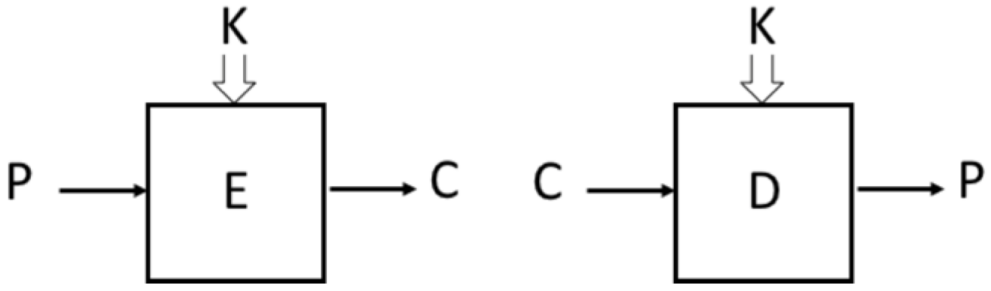

Figura 1.1: cifratura e decifratura

Inoltre, siano $\mathcal{P}$, $\mathcal{C}$ e $\mathcal{K}$ gli spazi, rispettivamente, del testo in chiaro P, del testo cifrato C e della chiave K. Presa una chiave $k \in \mathcal{K}$, gli algoritmi E_k e D_k sono delle applicazioni tra questi due spazi. Più precisamente,

$$E_k : \mathcal{P} \to \mathcal{C}$$
$$D_k : \mathcal{C} \to \mathcal{P}$$

Dal punto di vista più strettamente matematico, i sistemi crittografici si basano molto sull'aritmetica modulare, di cui diamo un'introduzione nel prossimo paragrafo. Nel successivo passeremo invece alla crittografia a chiave privata, con un occhio di riguardo ai problemi che hanno portato all'introduzione della crittografia a chiave pubblica (di cui parleremo nel quarto paragrafo). L'ultimo paragrafo sarà infine dedicato allo studio dell'RSA.

1.2 Aritmetica modulare

1.2.1 Quoziente e resto della divisione tra interi

Teorema 1 (della divisione di Euclide)**.** *Siano $a, b \in \mathbb{Z}$, con $b \neq 0$. Allora esistono e sono unici $q, r \in \mathbb{Z}$ tali che:*

$$a = bq + r \tag{1.1}$$

$$0 \leq r < |b| \tag{1.2}$$

I numeri a e b vengono chiamati, rispettivamente, *dividendo* e *divisore*, mentre q e r sono chiamati *quoziente* e *resto*. Il teorema è spiegato in Barile [2018].

Dimostrazione. Sia $b > 0$. Consideriamo l'insieme $Y = \{y > 0, y \in \mathbb{Z}, y = a - bx | x \in \mathbb{Z}\}$. Y non può essere vuoto; infatti:

- se $a \geq 0$, allora $0 \leq a = a - b \cdot 0 \in Y$
- se $a < 0$, dato che $b \geq 1$, allora $(1 - b)a \geq 0$, da cui segue $0 \leq (1 - b)a = a - ba \in Y$

Inoltre, $Y \subset \mathbb{N}$, quindi dall'assioma del buon ordinamento sappiamo che $\exists r \in \mathbb{N}$ tale che $r = \min Y$. Ma allora $\exists q \in \mathbb{Z}$ tale che $r = a - bq \geq 0$. Questo dimostra la (1.1) e la prima disuguaglianza di (1.2). Facciamo ora vedere che $r < |b| = b$. Per assurdo, poniamo $r \geq b$, da cui segue $0 \leq r - b = a - b(q + 1) \in Y$. Abbiamo quindi $r - b \in Y$ e $r - b < r$. Ma questo è assurdo, in quanto r è il minimo di Y; pertanto deve essere $r < |b|$.
Prendiamo invece $b < 0$, da cui ricaviamo immediatamente $-b > 0$; facendo gli stessi passaggi che abbiamo visto prima, troviamo $a = -bq + r = b(-q) + r$, con $0 \leq r < |b| = |-b|$
Fino qui abbiamo dimostrato l'esitenza; facciamo vedere anche l'unicità. Siano $q_1, q_2, r_1, r_2 \in \mathbb{Z}$ tali che $a = bq_1 + r_1$ e $a = bq_2 + r_2$, con $0 \leq r_1, r_2 < |b|$. Allora segue:

$$bq_1 + r_1 = bq_2 + r_2, \tag{1.3}$$

che implica $b(q_1 - q_2) = r_2 - r_1$. Se facciamo i valori assoluti e supponiano che $r_2 \geq r_1$:

$$|b||q_1 - q_2| = |r_2 - r_1| = r_2 - r_1 \leq r_2 < |b|, \tag{1.4}$$

da cui segue che $0 \leq |q_1 - q_2| < 1$; ma $q_1 - q_2 \in \mathbb{Z}$, quindi $|q_1 - q_2| = 0$, che vuol dire che necessariamente $q_1 = q_2$. Sostituendo in (1.3), otteniamo infine che $r_1 = r_2$. □

1.2.2 Congruenza in modulo

Questa sottosezione è tratta da Khan Academy e da Okalhoma State University.
Dati due interi a e b, con $b \neq 0$, abbiamo visto nel teorema 1 che esiste una e una sola coppia (q, r), tale che $a = bq + r$, con $q, r \in \mathbb{Z}$ e $0 \leq r < |b|$. Nell'ambito dell'aritmetica modulare si scrive $a \bmod b = r$ per indicare che il resto della divisione tra a e b è r. Ad esempio, se prendiamo $a = 13$ e $b = 5$, avremo che $q = 2$ e $r = 3$, da cui segue $13 \bmod 5 = 3$.
Dati due interi z_1 e z_2, diremo che sono **equivalenti in modulo r** se $z_1 \bmod r = z_2 \bmod r$ e scriveremo $z_1 = z_2 \pmod r$. Ad esempio, $26 = 11 \pmod 5$.
Fissato un intero positivo n, si verifica facilmente che il risultato dell'operazione $a \bmod n$ è un numero compreso tra 0 e $n - 1$, $\forall a \in \mathbb{Z}$. Un esempio immediato si ha con $n = 3$: $0 \bmod 3 = 0$, $1 \bmod 3 = 1$, $2 \bmod 3 = 2$, $3 \bmod 3 = 0$, e così via. L'insieme dei numeri da 0 a $n-1$ è chiamato insieme degli interi modulo n e si indica con $\mathbb{Z}_n$, come spiegato in McAndrew [2012]; in formula:

$$\mathbb{Z}_n = \{0,1,\dots,n-1\} \tag{1.5}$$

Dati due interi a ed n, chiameremo **classe di congruenza di a in modulo n** l'insieme di tutti gli interi x tali che $a = x \bmod n$, e la indicheremo con $[a]_n$. Ad esempio, sia $a = 3$ e $n = 5$; si verifica subito che $[3]_5 = \{3,3 \pm 5,3 \pm 10,3 \pm 15,\dots\}$

1.2.3 Addizione, sottrazione e moltiplicazione modulari

Siano $a, b, c \in \mathbb{Z}$, con $c > 0$. Come spiegato in Khan Academy, per l'addizione modulare vale la seguente proprietà:

Proprietà 1.

$$(a + b) \bmod c = (a \bmod c + b \bmod c) \bmod c \tag{1.6}$$

Prima di vedere la dimostrazione, fissiamo le idee con un esempio.

Esempio 1. *Sia $a = 17$, $b = 5$ e $c = 9$; nell'equazione (1.6) abbiamo:*

- *parte a sinistra dell'uguale:* $(17 + 5) \bmod 9 = 22 \bmod 9 = 4$
- *parte a destra dell'uguale:* $(17 \bmod 9 + 5 \bmod 9) \bmod 9 = (8 + 5) \bmod 9 = 13 \bmod 9 = 4$

Dimostrazione. Dal teorema 1 si ricava che $\exists (q_1, r_1), (q_2, r_2) \in \mathbb{Z}$ tali che

$$a = cq_1 + r_1 \tag{1.7}$$

$$b = cq_2 + r_2 \tag{1.8}$$

con $0 \leq r_1, r_2 \leq c$. Da (1.7) e (1.8) si ricava inoltre che:

$$a \bmod c = r_1 \tag{1.9}$$

$$b \bmod c = r_2 \tag{1.10}$$

In conclusione si ha quindi che

$$(a+b) \bmod c = (c(q_1+q_2)+r_1+r_2) \bmod c = (r_1+r_2) \bmod c = (a \bmod c+b \bmod c) \bmod c \tag{1.11}$$

dove per la prima uguaglianza si sono sommate (1.7) e (1.8); la seconda uguaglianza segue dalla definzione di modulo; la terza uguaglianza deriva invece da (1.9) e (1.10) □

Valgono delle proprietà analoghe anche per la sottrazione e la moltiplicazione modulari:

Proprietà 2.

$$(a - b) \bmod c = (a \bmod c - b \bmod c) \bmod c \tag{1.12}$$

Proprietà 3.

$$(a \times b) \bmod c = (a \bmod c \times b \bmod c) \bmod c \tag{1.13}$$

Le dimostrazioni di (1.12) e (1.13) sono analoghe a quella di (1.6) e vengono per questo motivo omesse.

1.2.4 Inverso modulare moltiplicativo

Tutte le sezioni dalla 1.2.4 alla 1.2.11 sono tratte da McAndrew [2012].

Definizione 1. *Dato un intero a, si definsisce* ***inverso moltiplicativo*** *in $\mathbb{Z}_n$ il numero $b \in \mathbb{Z}_n$ tale che $ab = 1 \pmod n$ e si scrive*

$$b = a^{-1} \pmod n$$

o, più raramente,

$$b = \frac{1}{a} \pmod n$$

Dati quindi a e n, per trovare a^{-1} si deve risolvere l'equazione

$$ax = 1 \pmod n \tag{1.14}$$

Il seguente teorema ci indica quando la soluzione di (1.14) esiste.

Teorema 2 (Esistenza dell'inverso)**.** *Dati due interi a ed n, l'inverso di a in $\mathbb{Z}_n$ esiste se e solo se a ed n sono coprimi, cioè se* $\operatorname{mcd}(a, b) = 1$

Dimostrazione. L'equazione (1.14) si può riscrivere come $ax = kn + 1$, da cui $ax - kn = 1$ con $k \in \mathbb{Z}$. Questa è una equazione diofantea lineare in due variabili che sappiamo (ad esempio da Wikipedia) avere soluzione intera se e solo se $\operatorname{mcd}(a, n) = 1$ □

Un modo computazionalmente efficiente per calcolare l'inverso sfrutta l'algoritmo di Euclide esteso e l'identità di Bézout. Della seconda diamo brevemente solo l'enunciato, mentre l'algoritmo di Euclide esteso verrà trattato in mamiera più esaustiva nella sezione 1.2.6

Teorema 3 (Identità di Bézout)**.** *Dati due interi m ed n, allora esistono altri interi s e t tali che:*

$$sm + tn = \operatorname{mcd}(m, n) \tag{1.15}$$

1.2.5 Algoritmo di Euclide

L'algoritmo di Euclide è usato per il calcolo del massimo comune divisore tra due interi m ed n. Il suo funzionamento si può così schematizzare:

1. calcolare r, resto della divisione intera tra m ed n: $r = m \bmod n$;
2. porre $m = n$ e $n = r$;
3. ripetere fino a quando $r = 0$; a quel punto si avrà che $\operatorname{mcd}(m, n) = n$

Dimostrazione. Osserviamo che l'algoritmo può essere descritto con questa sequenza di passi:

$$\begin{aligned} m &= q_1 n + r_1 \\ n &= q_2 r_1 + r_2 \\ r_1 &= q_3 r_2 + r_3 \\ r_2 &= q_4 r_3 + r_4 \\ &\dots \\ r_{k-2} &= q_k r_{k-1} + r_k \\ r_{k-1} &= q_{k+1} r_k \end{aligned} \tag{1.16}$$

dove l'ultimo resto, r_{k+1}, è uguale a 0, da cui segue che r_k è un divisore di r_{k-1}. Procedendo a ritroso, r_k è divisore anche di r_{k-2}, di r_{k-3}, e così via. r_k è anche divisore di m e di n, e quindi dell'mcd (m, n).
Facciamo ora vedere che r_k è proprio l'mcd (m, n). Sia g un altro intero che divide m ed n, cioè tale per cui si può scrivere: $m = xg$ e $n = yg$ per qualche $x, y \in \mathbb{Z}$. Dalle equazioni (1.16) ricaviamo che:

$$xg = m = q_1 n + r_1 = q_1 yg + r_1 \Leftrightarrow r_1 = xg - q_1 yg = (x - q_1 y)g = ag$$

ossia che g è un divisore anche di r_1.
Abbiamo anche che:

$$yg = n = q_2 ag + r_2 \Leftrightarrow r_2 = (y - q_2 a)g$$

cioè che g è un divisore anche di r_2. Iterando il ragionamento, arriviamo a dire che g è divisore di r_k, ma allora r_k deve per forza essere il mcd (m, n) □

Esempio 2. *Sia $m = 252$ e $n = 15$. Nella tabella 1.1 è sintetizzato il funzionamento dell'algoritmo di Euclide per il calcolo dell'* mcd (252,15)

i	m	n	r
1	252	15	12
2	15	12	3
3	12	3	0

Tabella 1.1: Algoritmo di Euclide

Essendo $r_3 = 0$, si ha: mcd (252,15) $= n_3 = 3$

1.2.6 Algoritmo di Euclide esteso

L'algoritmo di Euclide esteso si usa per calcolare due interi s e t che soddisfano l'identità di Bézout (1.15). Dati due interi m ed n, ad ogni step i l'algoritmo esegue i seguenti passaggi:

1. calcolo del resto i-esimo r_i; dalle formule viste per l'algoritmo di Euclide (1.16), si ricava che
$$r_i = r_{i-2} - r_{i-1} q_i \tag{1.17}$$
dove q_i è il quoziente della divisione intera tra r_{i-2} e r_{i-1}. Per calcolare r_1 si usano i seguenti valori iniziali:
$$\begin{cases} r_{-1} = m \\ r_0 = n \end{cases}$$

2. calcolo di due interi u_i e v_i tali che

$$mu_i + nv_i = r_i \tag{1.18}$$

Questi valori si possono ottenere facilmente dalle seguenti formule:

$$u_i = u_{i-2} - u_{i-1}q_i \tag{1.19}$$

$$v_i = v_{i-2} - v_{i-1}q_i \tag{1.20}$$

Per calcolare u_1 e v_1 si usano i seguenti valori iniziali:

$$\begin{cases} u_{-1} = 1 \\ v_{-1} = 0 \end{cases} \qquad \begin{cases} u_0 = 0 \\ v_0 = 1 \end{cases}$$

L'algoritmo si ferma allo step $i = k + 1$, per il quale si ha $r_{k+1} = 0$. Il valore r_k è l'mcd tra m ed n, metre u_k e v_k sono rispettivamente i valori s e t cercati.

Dimostrazione dell'algoritmo di Euclide esteso. Nella sezione 1.2.5 abbiamo dimostrato che r_k è l'mcd tra m ed n. Dobbiamo ancora dimostrare, però, che usando le equazioni (1.17), (1.19) e (1.20) si ottiene in automatico che (1.18) è valida. Questo si vede per induzione:

- se $i = 1$, abbiamo:

$$u_1 = u_{-1} - u_0q_1 = 1$$
$$v_1 = v_{-1} - v_0q_1 = -q_1$$
$$r_1 = r_{-1} - r_0q_1 = m - nq_1 = mu_1 + nv_1$$

- supponiamo ora che la relazione (1.18) sia valida $\forall j : 0 \leq j \leq i$. Facciamo vedere che è valida anche per $i + 1$.

$$r_{i+1} = r_{i-1} - r_iq_i = mu_{i-1} + nv_{i-1} - (mu_i + nv_i)q_i = m(u_{i-1} - u_iq_i) + n(v_{i-1} - v_iq_i) = mu_{i+1} + nv_{i+1}$$

dove la prima uguaglianza segue da (1.17), la seconda da (1.18) applicata a i ed $i - 1$, mentre per l'ultima si sono usate (1.19) e (1.20).

□

Esempio 3. *Sia $m = 45$ e $n = 70$. L'algoritmo di Euclide esteso esegue i calcoli sintetizzati nella tabella:*

i	q	r	u	v
−1		45	1	0
0		70	0	1
1	0	45	1	0
2	1	25	−1	1
3	1	20	2	−1
4	1	5	−3	2
5	4	0		

Tabella 1.2

Da cui si ricava che $s = -3$, $t = 2$ e $\text{mcd}\,(45{,}70) = 5$

1.2.7 Calcolo dell'inverso

Dati due numeri m ed n primi tra loro, applicando l'algoritmo di Euclide esteso si possono quindi ricavare s e t tali che

$$sm + tn = 1 \tag{1.21}$$

che si può riscrivere come

$$sm = (-t)n + 1$$

o come

$$tn = (-s)m + 1$$

Da queste equazioni segue che:

$$sm = 1(\mathrm{mod}n) \tag{1.22}$$

$$tn = 1(\mathrm{mod}m) \tag{1.23}$$

che significa che

$$s = m^{-1}(\mathrm{mod}n)$$

$$t = n^{-1}(\mathrm{mod}m)$$

Esempio 4. *Supponiamo di voler calcolare l'inverso di* 30 *in* $\mathbb{Z}_{47}$*. Poniamo* $m = 30$ *e* $n = 47$ *e eseguiamo l'algoritmo di Euclide esteso. Il risultato è* $s = 11$ *e* $t = -7$*, da cui* $30^{-1} = 11(\mathrm{mod}47)$*; inoltre,* $47^{-1} = -7 = 23(\mathrm{mod}30)$

1.2.8 Elevamento modulare a potenza

Per fare l'elevamento a potenza in modulo si sfrutta l'equazione (1.13) della proprietà della moltiplicazione modulare, da cui segue immediatamente che $a^2 \bmod n = (a \bmod n)^2$. L'obiettivo che si persegue è quello di non dover calcolare esplicitamente dei numeri molto grandi, ma solo il loro residuo in modulo. Per fare questo si può procedere in due modi:

- se l'esponente è una potenza di 2 ($esp = 2^k$), allora faccio l'elevamento al quadrato e ne calcolo il residuo, poi ripeto la stessa operazione ma con il residuo invece che con il numero di partenza, e così via, per un numero totale di k elevamenti al quadrato.
- se l'esponente non è una potenza di 2, allora ne scrivo la rappresentazione binaria; partendo dal bit più a sinistra, per ogni cifra binaria:
  - se è 1, elevo il risultato precedente al quadrato (1 se sono alla prima cifra), lo moltiplico per la base e ne faccio il modulo
  - se è 0, elevo solo il risultato precedente al quadrato e calcolo il modulo

  Esempio 5. *Per calcolare* $39^8 \bmod 43$*, osserivamo che* $8 = 2^3$*, quindi* $39^8 = (((39)^2)^2)^2$*, cioè* $k = 3$*. Si procede calcolando:*

  $$39^2 = 1521 = 16 \bmod 42$$

  $$16^2 = 256 = 4 \bmod 42$$

  $$4^2 = 16 = 16 \bmod 42$$

  Perciò $39^8 = 16(\bmod 42)$*. Osserviamo anche che il numero più grande che abbiamo dovuto calcolare è stato* 1521*, mentre se avessimo calcolato direttamente* 39^8 *avremmo ottenuto* 5352009260481

Esempio 6. *Calcoliamo* $39^{25} \bmod 53$. *La rapresentazione binaria di* 25 *è* 11001*; partendo dal primo bit a sinistra si ha*

- *1*$\rightarrow 1^2 \times 39 = 39 = 39(\text{mod}53)$
- *1*$\rightarrow 39^2 \times 39 = 59319 = 12(\text{mod}53)$
- *0*$\rightarrow 12^2 = 144 = 38(\text{mod}53)$
- *0*$\rightarrow 38^2 = 1444 = 13(\text{mod}53)$
- *1*$\rightarrow 13^2 \times 39 = 6591 = 19(\text{mod}53)$

Quindi il risultato finale sarà $39^{25} = 19(\text{mod}53)$. *In questo caso il numero più grande è* 59319*, mentre se avessimo calcolato direttamente* 39^{25} *avremmo ottenuto* 597881582915616404980520276877467472099 9.

1.2.9 Teorema cinese del resto

Teorema 4 (cinese del resto)**.** *Siano* m *ed* n *due interi primi tra loro. Allora esiste una singola* x *soluzione in* $\text{mod}(nm)$ *alle congruenze:*

$$x = a(\text{mod}m)$$

$$x = b(\text{mod}n)$$

Dimostrazione. Abbiamo visto che se m e n sono primi tra loro, si può usare l'algoritmo di Euclide esteso per calcolare s e t tali che $sm + tn = 1$. Inoltre, poichè valgono (1.22) e (1.23), prendiamo $x = bsm + atn$. Questa x soddisfa entrambe le congruenze, infatti:

$$x \bmod m = atn \bmod m = a$$

$$x \bmod n = bsm \bmod n = b$$

□

Esempio 7. *Cosideriamo la doppia congruenza*

$$x = 4(\text{mod}5)$$

$$x = 3(\text{mod}7)$$

Siccome $n = 5$ *e* $m = 7$ *sono due numeri primi relativi, è possibile applicare l'algoritmo di Euclide esteso per calcolare* t *ed* s*, come indicaro in tabella 1.3*

i	q	r	u	v
-1		7	1	0
0		5	0	1
1	1	2	1	-1
2	2	1	-2	3
3	2	0	5	-7

Tabella 1.3: Algoritmo di Euclide esteso per il calcolo dell'inverso

Poichè $s = -2$ *e* $t = 3$*, si avrà*

$$x = bsm + atn = 4(-2)7 + 3(3)5 = -11 = 24(\text{mod}35)$$

1.2.10 Funzione di Eulero

Definizione 2. *Sia n un numero intero positivo e sia $R(n)$ l'insieme dato da:*

$$R(n) = \{x : 0 < r < n, \mathrm{mcd}\,(r, n) = 1\}$$

Si definisce ***funzione*** ϕ ***di Eulero*** *la cardinalità di $R(n)$ e si indica con $\phi(n)$.*

In altre parole, $\phi(n)$ è il numero degli interi più piccoli di n e ad esso coprimi.

Esempio 8. *Sia $n = 15$. In questo caso*

$$R(15) = \{1,2,4,7,8,11,13,14\}$$

da cui $\phi(15) = |R(15)| = 8$

Vediamo due proprietà della funzione ϕ di Eulero che useremo più avanti.

Proprietà 4. *Se p è un numero primo, allora $R(p) = \{1,2,3,\ldots,p-1\}$, da cui $\phi(p) = |R(p)| = p - 1$*

Proprietà 5. *Se m, n sono coprimi, $\phi(mn) = \phi(m)\phi(n)$.*
Inoltre, se p, q sono primi, $\phi(pq) = \phi(p)\phi(q) = (p-1)(q-1)$

Per la funzione ϕ vale anche il seguente teorema:

Teorema 5 (di Eulero)**.** *Siano a, n due interi coprimi. Allora:*

$$a^{\phi(n)} = 1 \bmod n \tag{1.24}$$

Dimostrazione. Sia $R = \{r_1, r_2, \ldots, r_{\phi(\mathrm{n})}\}$. Facciamo vedere che:

$$r_1 a \neq r_2 a, \ldots, \neq r_{\phi(\mathrm{n})} a (\mathrm{mod} n) \tag{1.25}$$

Per assurdo, se $r_\mathrm{i} a = r_\mathrm{j} a$ per qualche i, j, allora

$$r_\mathrm{i} a - r_\mathrm{j} a = (r_\mathrm{i} - r_\mathrm{j}) a = 0 (\mathrm{mod} n)$$

Questo implica che o a o $r_\mathrm{i} - r_\mathrm{j}$ è divisibile per n. La prima ipotesi è da scartare, in quanto a ed n sono coprimi; la seconda invece è valida se e solo se $r_\mathrm{i} - r_\mathrm{j} = 0$, cioè se $r_\mathrm{i} = r_\mathrm{j}$, ma anche questo è impossibile, perchè gli r_k sono diversi tra loro al variare di k. Dunque abbiamo trovato la contraddizione per cui possiamo dire che (1.25) è vera.
Dalla (1.25) ricaviamo anche che i valori $r_1 a, r_2 a, \ldots, r_{\phi(\mathrm{n})} a (\mathrm{mod} n)$ sono tutti e soli i valori $r_1, r_2, \ldots, r_{\phi(\mathrm{n})} a$ in qualche ordine. Allora segue che:

$$(r_1 a)(r_2 a) \ldots (r_{\phi(\mathrm{n})} a) (\mathrm{mod} n) = r_1 r_2 r_{\phi(\mathrm{n})} (\mathrm{mod} n)$$

Siccome il prodotto $r_1 r_2 \ldots r_{\phi(\mathrm{n})}$ non è divisiblie per n, possiamo dividere a destra e a sinistra per questa quantità, ottenendo così l'enuciato del teorema:

$$a^{\phi(n)} = 1 \bmod n$$

□

1.2.11 Gruppo, anello e campo finito

$\mathbb{Z}_n$ come gruppo

L'insieme $\mathbb{Z}_n$ è un *gruppo abeliano* rispetto all'addizione in modulo n (o **gruppo abeliano additivo**); infatti valgono le segueti proprietà:

- **chiusura**: $\forall a, b \in \mathbb{Z}_n$, $(a + b) \bmod n \in \mathbb{Z}_n$
- **associatività**: $\forall a, b, c \in \mathbb{Z}_n$, $((a + b) + c) \bmod n = (a + (b + c)) \bmod n$
- **esistenza dell'identità**: $\exists e \in \mathbb{Z}_n$ tale che $\forall a \in \mathbb{Z}_n$, $(a + e) \bmod n = (e + a) \bmod n = a$. In questo caso $e = 0$
- **esistenza dell'inverso**: $\forall a \in \mathbb{Z}_n$, $\exists b \in \mathbb{Z}_n$ tale che $(a + b) \bmod n = 0$ e si scrive $b = -a$
- **commutatività**: $\forall a, b \in \mathbb{Z}_n$, $(a + b) \bmod n = (b + a) \bmod n$

Per comodità di notazione, indichamo con $+$ l'operazione di addizione in modulo n: $a+b = a+b \pmod n$.
Usando delle apposite tabelle, è facile verifcare le proprietà che abbiamo descritto per i gruppi $\mathbb{Z}_n$. Nella tabella 1.4 è mostrato l'esempio per $\mathbb{Z}_4$

$+$	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

Tabella 1.4: Il gruppo $\mathbb{Z}_4$

Diamo adesso due definizioni che useremo più avanti: quella di gruppo ciclico e di generatore

Definizione 3. *Un gruppo additivo G è detto **ciclico** se esiste un elemento $g \in G$, $g \neq e$, tale che la successione*

$$g$$
$$g + g = 2 \cdot g$$
$$g + g + g = 3 \cdot g$$

*e così via, genera tutto G. Tale elemento g è chiamato **generatore**.*

Inoltre, se G è finito, $\exists m$ tale che $m \cdot g = \underbrace{g + g + \cdots + g}_{\text{m volte}} = e$ (identità).

Esempio 9. *Nel caso di $\mathbb{Z}_n$, con n qualsisasi, $g = 1$; infatti:*

$$1$$
$$2 \cdot 1 = 1 + 1 = 2$$
$$3 \cdot 1 = 1 + 1 + 1 = 3$$
$$\cdots$$
$$(n-1) \cdot 1 = \underbrace{1 + 1 + 1 + \cdots + 1}_{\text{n-1 volte}} = n - 1$$
$$n \cdot 1 = \underbrace{1 + 1 + 1 + \cdots + 1}_{\text{n volte}} = 0$$

$\mathbb{Z}_n$ come anello

Il concetto di **anello** è una generalizzazione del concetto di gruppo.

Definizione 4. *Per anello con identità si intende un insieme R dotato di due operazioni, indicate con $+$(addizione) e $\times$ (moltiplicazione), e di due elementi distiniti 0 e 1 tali che:*

- *la coppia $(R, +)$ è un gruppo abeliano, con 0 come identità;*
- *$\forall x, y \in R$, l'operazione $x \times y$ soddisfa le proprietà di chiusura e di associatività su R*
- *l'elemento 1 è l'identità rispetto alla moltiplicazione, cioè $x \times 1 = 1 \times x = x$, $\forall x \in R$*
- *la moltiplicazione è distributiva rispetto all'addizione:*

$$x \times (y + z) = (x \times y) + (x \times z)$$

$$(x + y) \times z = (x \times z) + (y \times z)$$

$\forall x, y, z \in R$

Dalla definizione segue immediatamente che $\mathbb{Z}_n$ è un anello

$\mathbb{Z}_n$ come campo finito

Definizione 5. *Un campo è un anello R per cui sia $(R, +)$ sia $(R, \times)$ sono gruppi abeliani. Più precisamente, un campo è un insieme $\mathbb{F}$ tale per cui, $\forall x, y, z \in \mathbb{F}$,:*

- *l'addizione e la moltiplicazione sono chiuse: $x + y \in \mathbb{F}$ e $x \times y \in \mathbb{F}$*
- *l'addizione e la moltiplicazione sono commutative e associative: $x+y = y+x$, $x \times y = y \times x$; $(x + y) + z = x + (y + z)$, $x \times y \times z = (x \times y) \times z$*
- *esistono l'identità additiva (0) e quella moltiplicativa (1): $x+0 = 0+x = x$, $x \times 1 = 1 \times x = x$*
- *esistono l'inverso additivo $-x$ e, se $x \neq 0$, quello moltiplicativo x^{-1}: $x + (-x) = 0$, $x \times x^{-1} = 1$*
- *l'addizione è distributiva rispetto alla moltiplicazione: $x \times (y + z) = x \times y + x \times z$*

Osservazione 1. *Il fatto che la definzione di campo richieda la presenza di un inverso moltiplicativo, ci permette di affermare che $\mathbb{Z}_n$ è un anello solo nel caso in cui n è un numero primo; infatti se n non è un numero primo, allora $\exists a \in \mathbb{Z}_n$ non coprimo con n, per cui non è possibile calcolare l'inverso moltplicativo.*

1.3 Crittografia a chiave privata

Nella crittografia a chiave privata, la chiave k usata per criptare e la chiave h usata per decriptare sono uguali, o comunque data una si può risalire facilmente all'altra Caressa [2005]. In questo sistema mittente e destinatario devono trovare un modo per scambarsi la chiave, senza che questa venga intercettata, prima di iniziare la trasmissione.

Esempio 10. *Un classico esempio di cifrario a chiave privata è il cosiddetto Cifrario di Cesare McAndrew [2012]. Supponiamo che due persone, che usano un certo alfabeto, decidamo di comunicare usando questo cifrario. Dopo essersi messe d'accordo sulla chiave k, procedono in questo modo:*

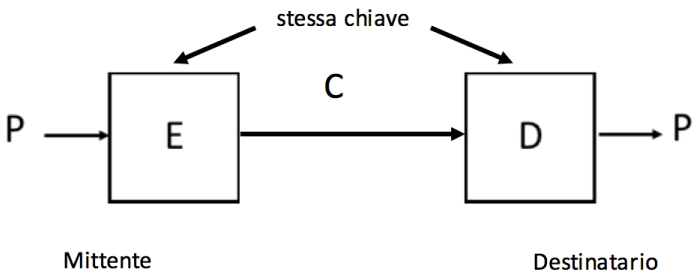

Figura 1.2: crittografia a chiave privata

- *il mittente sposta in avanti ogni lettera del messaggio di k posizioni, con la regola che dopo la Z l'alfabeto ricomincia dalla A, e spedisce il messaggio;*
- *il destinatario sposta all'indietro ogni lettera del messaggio di k posizioni, con la regola che prima della A viene la Z*

Per esempio, se $k = 3$ e se prendiamo l'alfabeto italiano, ogni lettera viene trasformata come mostrato nella tabella 1.5. Il messaggio PRELIEVO DI CENTO EURO risulterebbe in questo

A	B	C	D	E	F	G	H	I	L	M	N	O	P	Q	R	S	T	U	V	Z
D	E	F	G	H	I	L	M	N	O	P	Q	R	S	T	U	V	Z	A	B	C

Tabella 1.5: Cifrario di Cesare con $k = 3$

modo cifrato come SUHONUBR GN FHQZR HAUR; per decifrarlo, il destinatario non deve far altro che usare la stessa tabella ma al contrario, partendo dalla seconda riga e sostituendo le lettere con quelle corrispondenti della prima riga.
Dal punto di vista matematico, il procedimento si può cosi formalizzare:

- *ad ogni lettera viene associato un numero in $\mathbb{Z}_{n-1}$, dove n è il numero di lettere dell'alfabeto usato (per l'alfabeto italiano, $n = 21$ e $A = 0, B = 1, \ldots, Z = 20$);*
- *per cifrare si somma ogni numero con la chiave k in $\mathbb{Z}_{n-1}$: $c = p + k \bmod (n-1)$, dove p rappresenta il testo (numerico) in chiaro, mentre c è il testo cifrato;*
- *per decifrare si fa l'operazione inversa, ossia la sottrazione in $\mathbb{Z}_{n-1}$: $p = c - k \bmod (n-1)$*

1.3.1 Problemi della crittografia a chiave privata

Il problema principale, come evidenziato in Caressa [2005], sta nel fatto che in un sistema di comunicazione con crittogafia a chiave privata, il numero di chiavi necessarie aumenta con il numero di utenti. Se ad esempio abbiamo n utenti, e ognuno di loro può comunicare con tutti gli altri, allora il numero di chiavi necessario sarà $n(n-1)/2$, ossia il numero di lati di un grafo completo indiretto con n vertici. In generale, per n grande, il numero di chiavi sarà $\approx n^2$. Un altro problema è quello evidenziato nel famoso articolo di Diffie e Helmman Diffie and Hellman [1976], ossia il cosiddetto *key exchange*, scambio di chiavi. Per usare un sistema a chiave privata, occorre che due utenti che vogliono comunicare trovino un modo di scambiarsi la chiave attraverso un canale sicuro, come può essere ad esempio un corriere o un canale di posta registrata. Tuttavia,

esistono dei casi in cui due persone che non si conoscono devono comunicare tra loro in modo sicuro, ma non hanno il tempo di scambiarsi informazioni attraverso un mezzo fisico; pensiamo, per esempio, al caso di contatti per affari.
Per questi motivi nasce la crittografia a *chiave pubblica*, in cui, partendo dal presupposto che il canale di trasmissione non è sicuro, ogni utente genera una coppia di chiavi e ne tramette solo una. Questa seconda chiave è conosciuta da tutti, ma questo non incide sulla sicurezza del sistema, come vedremo meglio nella prossima sezione.

1.4 Crittografia a chiave pubblica

Nei sistemi a chiave pubblica ciascun utente genera due chiavi: una pubblica e una privata. La chiave privata è nota solo a chi l'ha generata (non viene trasmessa sul canale), mentre la chiave pubblica è conosciuta da tutti. Quando si vuole mandare un messaggio, lo si cifra con la chiave pubblica del destinatario, e il sistema è costruito in modo che solo lui lo possa decriptare usando la sua chiave privata. Per fare questo si possono usare, ad esempio, le cosiddette funzioni *one-way* Weisstein, cioè quelle funzioni f facili da calcolare ma difficili da invertire: dato x è facile calcolare $y = f(x)$, ma dato $y = f(x)$, è difficile ricavare x, a meno di non trovarsi in condizioni particolari. Un tipico esempio di funzione one-way è il seguente:

Esempio 11. *Siano $n, a \in \mathbb{Z}$ fissati. Abbiamo visto nella sezione 1.2.8 che, $\forall x \in \mathbb{Z}$, il calcolo di $f(x) = a^x \pmod n$ è facile da fare. Il calcolo inverso, cioè data $f(x)$ trovare x tale che $a^x \bmod n = f(x)$ (problema del **logaritmo discreto**), è difficile, nel senso che non esiste (per ora) un algoritmo in grado di eseguirlo in modo efficiente McAndrew [2012].*

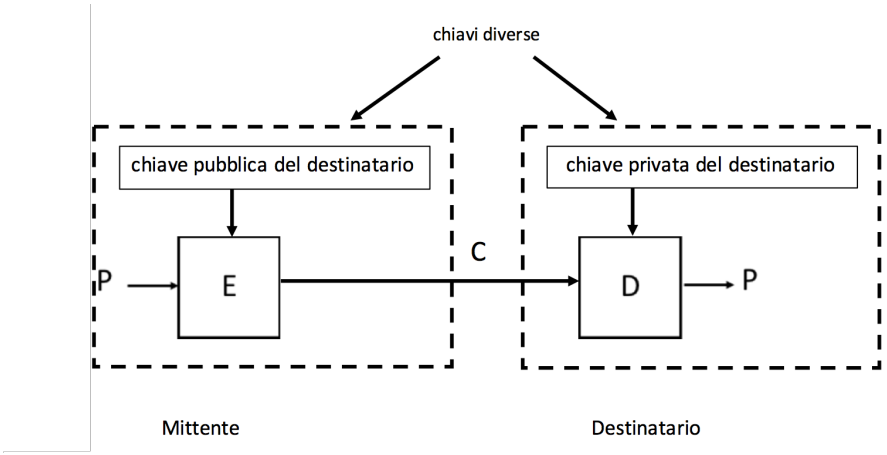

Figura 1.3: crittografia a chiave pubblica

Osservazione 2. *Quando diciamo che le funzioni one-way sono facili da calcolare e difficili da invertire, intendiamo dire che, **allo stato attuale**, esistono degli algoritmi in grado di calcolarle in modo computazionalmente efficiente, ma non ne esistono in grado di invertirle, a meno di non trovarci in una situazione particolare. Se per caso venisse scoperto un algoritmo di inversione efficiente, la funzione corrispondente cesserebbe di essere one-way e l'eventuale sistema crittografico basato su di essa sarebbe irrimediabilmente rotto.*

Dal punto di vista formale, ricaviamo la definizione di un sistema a chiave pubblica dall'articolo di Diffie-Helmann Diffie and Hellman [1976]. Riprendiamo la notazione introdotta nella sezione 1.1, assumendo inoltre che lo spazio dei testi in chiaro $\mathcal{P}$ sia lo stesso di quelli cifrati $\mathcal{C}$, che indichiamo con $\mathcal{M}$, spazio dei messaggi:

$$\mathcal{P} \equiv \mathcal{C} \triangleq \mathcal{M}$$

In questo modo gli algoritmi di cifratura E_k e di decifratura D_k sono delle applicazioni da $\mathcal{M}$ in sè stesso $\forall k \in \mathcal{K}$:

$$E_k : \mathcal{M} \to \mathcal{M}$$
$$D_k : \mathcal{M} \to \mathcal{M}$$

In un sistema a chiave pubblica, gli algoritmi E_k e D_k devono inoltre soddisfare le seguenti proprietà:

1. $\forall k \in \mathcal{K}$, E_k è l'inverso di D_k:
$$D_k(E_k(m)) = m$$
$\forall m \in \mathcal{M}$
2. $\forall k \in \mathcal{K}$ e $\forall m \in \mathcal{M}$, E_k e D_k sono facili da calcolare (nel senso descritto nell'osservazione 2).
3. per quasi ogni $k \in \mathcal{K}$, avere a disposizione l'algoritmo di cifratura E_k non implica essere in grado di ricavare il corrispondente algoritmo di decifraura D_k; in altre parole, il calcolo di D_k a partire da E_k è infattibile computazionalmente. Questo garantisce che rendendo pubblica la chiave di cifratura non viene compromessa la sicurezza della chiave di decifratura
4. $\forall k \in \mathcal{K}$, è sempre possibile calcolare una coppia di trasformazioni inverse E_k, D_k

Vediamo ora un esempio che, sebbene non sia un sistema crittografico a chiave privata in senso stretto, può essere utile a fissare le idee. Anche questo esempio è tratto da Diffie and Hellman [1976].

Esempio 12. *Consideriamo il caso in cui un certo numero di utenti voglia scambiarsi delle email. Dato il campo $\mathbb{Z}_p$, dove p è un numero primo, ciascun utente i sceglie un numero x_i senza rivelarlo agli altri (chiave privata) e calcola α^{x_i}, dove α è un generatore di $\mathbb{Z}_p$; il valore α^{x_i} è reso pubblico.*

Quando due utenti i e j vogliono scambiarsi le mail, l'utente i cifra usando la chiave $k_{ij} = \alpha^{x_j^{x_i}}$, cioè sfrutta la sua chiave privata e la chiave pubblica di j. Per decifrare, l'utente j calcola $k_{ji} = \alpha^{x_i^{x_j}}$, sfruttando anche lui la sua chiave privata e la chiave pubblica di i. Vediamo in maniera immediata che $k_{ij} = k_{ji}$, per cui i due utenti si possono scambare informazioni.

In un sistema a chiave pubblica il numero totale di chiavi diminuisce notevolmente: infatti, se n è il numero totale di utenti, ognuno di loro avrà la sua coppia di chiavi (pubblica e privata), per un totale di $2n$ chiavi.

1.5 Sistema crittografico RSA

Per questa sezione, è stato usato il materiale di McAndrew [2012].
Il sistema crittografico RSA viene inventato nel 1977 da Ronald Rivest, Adi Shamir, e Leonard Adleman, dalle cui iniziali prende il nome. Nella tabella 1.6 ne viene data una breve descrizione.

Parametri	due interi grandi p e q ed il loro prodotto $n = pq$
Chiave pubblica	la coppia (n, e), con e coprimo di $\phi(n)$, $e < n$
Chiave privata	la terna (d, p, q), con $d = e^{-1} \bmod (p-1)(q-1)$
Cifratura	dato un messaggio $m < n$, il testo cifrato è $c = m^e \bmod n$
Decifratura	dato un messaggio cifrato c, il testo in chiaro è $m = c^d \bmod n$

Tabella 1.6: sistema crittografico RSA

1.5.1 Funzionamento di RSA nel dettaglio

Prima di tutto occorre prendere due interi p e q abbastanza grandi (che in questo contesto significa almeno settecento cifre binarie ciascuno) e calcolarne il prodotto:

$$n = pq$$

A questo punto si sceglie un qualunque intero e che sia coprimo a $(p-1)$ e $(q-1)$, in modo tale che esista l'inverso $d = e^{-1} \pmod{(p-1)(q-1)}$, per calcolare il quale è possibile usare l'algoritmo di Euclide esteso. La coppia (e, n) costituisce la chiave pubblica, mentre la terna (d, p, q) è la chiave privata.

- per **cifrare**, si divide il testo numerico in blocchi, facendo in modo che ogni blocco m sia più piccolo di n $(m < n)$ e si calcola il crittogramma c come $c = m^e (\mathrm{mod} n)$
- per **decriptare**, si calcola $m = c^d (\mathrm{mod} n)$

Facciamo vedere che vale il seguente risulato: $c^d = m \pmod{n}$. Per fare questo distinguiamo due casi:

1. m è relativamente primo a p e q, e conseguentemente anche a n
2. m è multiplo di uno dei due, ad esempio di p

Osserviamo che m non può essere multiplo di entrambi, in quanto se così fosse sarebbe anche multiplo di n, per cui si avrebbe $m \geq n$, che non è possibile.
Nel caso 1 si ha:

$$c^d = (m^e)^d (\mathrm{mod} n) \tag{1.26}$$

Sappiamo che e e d sono inversi in $\mathrm{mod}(p-1)(q-1)$, cioè che

$$ed = 1 \bmod (p-1)(q-1)$$

da cui segue che

$$ed = k(p-1)(q-1) + 1$$

per qualche $k \in \mathbb{Z}$. Quindi

$$m^{ed} = mm^{k(p-1)(q-1)} = m(m^{\phi(n)})^k,$$

dove $\phi(n)$ è la ϕ di Eulero: $\phi(n) = \phi(pq) = (p-1)(q-1)$ perchè p e q sono numeri primi. Ma siccome m ed n sono numeri coprimi, sappiamo dal teorema di Eulero che $m^{\phi(n)} = 1(\mathrm{mod} n)$. Riprendendo quindi (1.26) si ha:

$$c^d = m^{e^d} (\mathrm{mod} n) = mm^{\phi(n)^k} (\mathrm{mod} n) = m(\mathrm{mod} n)$$

Nel caso 2 si ha che $m = kp$ per qualche $k \in \mathbb{Z}$, da cui segue:

$$m^{ed} = (kp)^{ed} = 0(\mathrm{mod}p) \tag{1.27}$$

$$m = kp = 0(\mathrm{mod}p) \tag{1.28}$$

Mettendo insieme (1.27) e (1.28) si ricava che

$$m^{ed} = m(\mathrm{mod}p) \tag{1.29}$$

Inoltre,

$$m^{ed} = mm^{(q-1)k(p-1)}$$

Siccome m è coprimo a q, applicando di nuovo il teorema di Eulero, ricaviamo

$$m^{(q-1)} = 1(\mathrm{mod}p)$$

da cui segue che

$$m^{ed} = m(\mathrm{mod}q) \tag{1.30}$$

In conclusione, usando il teorema cinese del resto con (1.29) e (1.30), arrivo al risultato finale:

$$m^{ed} = m(\mathrm{mod}pq)$$

Proproniamo ora un esempio con numeri molto bassi, unicamente per fissare le idee, nella realtà si usano cifre più alte, come quelle riportate nella tabella 1.7

RSA-100	1522605027922533360535618378132637429718068114961380688657908494580122963258952897654000350692006139
RSA-200	27997833911221327870829467638722601621070446786955428537560009929326128400107609345671052955360856061822351910951365788637105954482006576775098580557613579098734950144178863178946295187237869221823983
RSA-250	21403246502407449612644230728393335630086147151447550177977549208814180234471401366433455190958046796109928518724709145876873962619215573630474547705208051190564931066876915900197594056934574522305893259766974716817380693648946998715784949759374979937

Tabella 1.7: esempi di numeri RSA, rispettivamente con 100, 200 e 250 cifre, tratta da Numeri RSA - Wikipedia

Esempio 13. *Siano* $p = 13$ *e* $q = 23$*, da cui*

$$n = 13 \times 23 = 299$$

Per l'altro numero della chiave privata e*, scegliamo un valore che sia coprimo di* $(p-1)(q-1) = 264$*, per esempio* $e = 17$*. La chiave privata* d *sarà quindi data da:*

$$d = e^{-1}(\mathrm{mod}264) = 233$$

Supponiamo che il messaggio da trasmettere sia stato codificato in

$$27622312743$$

Tale messaggio può essere spezzato in blocchi da 3 cifre ciascuno: $m_1 = 276$, $m_2 = 223$, $m_3 = 127$, $m_4 = 47$. *Per ogni blocco* m_i *il mittente calcola* $c_\text{i} = m_\text{i}^e \bmod n$*:*

$$c_1 = 276^{17} \bmod 299 = 230$$

$$c_2 = 223^{17} \bmod 299 = 188$$

$$c_3 = 127^{17} \bmod 299 = 147$$

$$c_4 = 47^{17} \bmod 299 = 47$$

Per decrittare, il destinatario calcola $m_\text{i} = c_\text{i}^d (\text{mod} n)$

$$m_1 = 230^{233} \bmod 299 = 276$$

$$m_2 = 188^{233} \bmod 299 = 223$$

$$m_3 = 147^{233} \bmod 299 = 127$$

$$m_4 = 47^{233} \bmod 299 = 47$$

1.5.2 Sicurezza

Per rompere un sistema RSA occorre risolvere o il problema di fattorizzazione di un intero (e in questo caso si ricava la chiave privata (d, p, q)) o il problema della radice quadrata discreta (e in questo caso si ha il messaggio m). Entrambi sono però difficili da risolvere con gli algoritmi noti. Questo non significa che abbiamo la certezza assoluta che RSA sia sicuro, ma ci permette comunque di usarlo con un buon grado di confidenza, almeno fino a quando non saranno trovati algoritmi efficienti per romperlo (a questo proposito, si rimanda ai capitoli successivi, in particolare al 2).

Problema della fattorizzazione: calcolo della chiave privata (d, p, q)

Data la chiave pubblica (e, n), per calcolare l'esponente di decriptazione d occorre conoscere $\phi(pq) = (p-1)(q-1)$, che a sua volta richiede di conoscere i valori p e q, cioè di fattorizzare n. Ma sappiamo che tale problema è infattibile computazionalmenete, in particolar modo per n molto grandi. Se ad esempio si prende n con 200 cifre, che è il prodotto di due primi di 100 cifre, la sua fattorizzazione richiede un tempo superiore a quello dell'età dell'universo! Occorre comunque precisare che, se n e $\phi(n)$ sono noti, allora p e q sono facili da calcolare. Infatti:

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

da cui segue che

$$p + q = n - \phi(n) + 1$$

Queste formule fanno vedere che dati n e $\phi(n)$, abbiamo anche la somma di p e q e il loro prodotto, da cui poi si risale facilmente ai loro valori.

Problema della radice discreta: ricostruzione del messaggio m

Dati c, n ed e, è lecito chiedersi se è possibile risalire al messaggio m tale che $m = c^e \bmod n$. Questo problema è noto come *radice discreta*, in analogia con l'operazione di estrazione della radice in $\mathbb{Z}$. Sappiamo che quello della radice discreta è un problema difficile; uno degli algortimi per risolverlo, ad esempio, richiede la conoscenza di un generatore di $\mathbb{Z}_n$ e il calcolo di un logaritmo discreto, che sappiamo essere entrambi dei problemi infattibili computazionalmente. L'algoritmo è dettagliato in Discrete Root - algorithms.

Problema del computer quantistico: rottura di RSA

Quello del computer quantistico è, ad oggi, il problema più stringente da affrontare: esiste infatti un algoritmo di tipo quantistico che è in grado di fattorizzare un intero n in un tempo polinomiale, determinando in questo modo la rottura totale di RSA. Al netto di non sapere ancora con precisione se e quando i computer quantistci verranno commercializzati, sono in fase di studio ormai avanzata alcuni sistemi crittografici resistenti agli attacchi da parte di tali computer. Analizzeremo questi argomeni più in dettaglio nei prossimi capitoli.

1.5.3 Attacchi ad un sistema RSA

Per questa sottosezione sono stati usati il libro McAndrew [2012] e l'articolo Boneh [1999]

Attacco a modulo comune

Per descrivere questo attacco, dobbiamo usare il seguente teorema:

Teorema 6. *Noti d, n ed e in un sistema RSA, si può ricavare facilmente la fattorizzazione di n.*

Dimostrazione. Dati d ed e, calcoliamo $k = de - 1$. Dalle definizioni di d e e, segue che k è multiplo di $\phi(n)$, e possiamo scrivere

$$k = l\phi(n)$$

per qualche $l \in \mathbb{Z}$. Questo implica che k è un numero pari, e quindi si può scrivere come $k = 2^t r$, con r dispari e $t > 1$.
Inoltre, preso un messaggio m qualunque, si ha che

$$m^k (\mathrm{mod} n) = m^{l\phi(n)} (\mathrm{mod} n) = 1$$

(i passaggi in dettaglio si trovano nella sezione 1.5.1). Ma questo vuol dire che $m^{k/2} = \sqrt{1} (\mathrm{mod}\ n)$. Usando una generalizzazione del teorema cinese del resto, abbiamo che 1 ha quattro radici quadrate in mod$n$: ± 1 e $\pm x$, dove x è tale che

$$x = 1 (\mathrm{mod} p)$$

e

$$x = -1 (\mathrm{mod} q)$$

Possiamo quindi fattorizzare n calcolando il mcd $(n, x - 1)$. □

Supponiamo di avere un sistema RSA anche solo con due utenti, A e B. Questi due utenti usano lo stesso n, ma diversi d ed e:
Un messaggio cifrato diretto ad A sarà quindi nella forma $c = m^{e_a}$. L'utente B, però, conoscendo e_a, n e d_b, sfruttando il risultato che abbiamo dimostrato, può fattorizzare n e quindi decriptare il messaggio.

Utente	chiave pubblica	chiave privata
A	(n, e_a)	d_a
B	(n, e_b)	d_b

Tabella 1.8: Esempio di sistema RSA in cui due utenti usano lo stesso n

Attacco 'low encryption'

Quando lo stesso messaggio m viene cifrato più volte con lo stesso esponente e (piccolo) ma usando valori diversi di n, il Teorema Cinese del Resto si può usare per ricavare m.

Esempio 14. *Consideriamo tre valori di n:* $n_1 = 1591$, $n_2 = 1927$, $n_3 = 1643$. *Prendiamo* $m = 500$ *e* $e = 3$ *e cifriamo usando i tre valori di n. Otteniamo:*

$$c_1 = m^3 \bmod 1591 = 1494$$

$$c_2 = m^3 \bmod 1927 = 1291$$

$$c_3 = m^3 \bmod 1643 = 560$$

che può essere visto come un sistema di equazioni:

$$m^3 \bmod 1591 = 1494$$

$$m^3 \bmod 1927 = 1291$$

$$m^3 \bmod 1643 = 560$$

Applicando il teorema cinese del resto, si ha $m^3 = 125.000.000$, *da cui* $m = 500$

Attaco a tempo

P. Kocher dimostra nel 1996 che un attacco a tempo permette di calcolare la chiave privata d. Sfruttando il fatto che nell'algoritmo di elevamento a potenza modulare alcuni passi richiedono più tempo di altri, analizzando con precisione la velocità di tali algoritmi si può risalire alla chiave privata. Questo attacco si rivela essere pratico, ma può essere contrastato usando la cosiddetta strategia di *blinding*:

- Generare un numero r random compreso tra 0 e $n = pq$
- Calcolare $c' = cr^e (\mathrm{mod} n)$, dove e è parte della chiave pubblica
- Calcolare $m' = (e')^d (\mathrm{mod} n)$
- Ricostruire il testo in chiaro $m = m' r^{-1} (\mathrm{mod} n)$

Osserviamo che il procedimento funziona, in quanto $r^{ed} = r (\mathrm{mod} n)$.

Capitolo 2

Algoritmo di Shor

2.1 Introduzione

Dal punto di vista teorico, quello del quantum computing è un ambito di ricerca molto affascinante: i qubit, ossia le unità fondamentali di memoria dei computer quantistici, possono infatti esistere contemporaneamente in una sovrapposizione di stati, rendendo in questo modo fattibili operazioni anche molto complesse. Ed è proprio su questa proprietà che si basa l'algoritmo di Shor, scoperto nel 1994 ed in grado, come abbiamo già accennato, di mettere in crisi il sistema RSA.
Nonostante la prima intuizione sul quantum computing risalga al 1959 Quantum Computing-Wikipedia, è proprio l'algoritmo di Shor a dare un'accelerata al suo sviluppo, mostrandone per la prima volta le ricadute pratiche. In parallelo, aumenta anche la ricerca sui sistemi crittografici ritenuti sicuri in caso di attacco con gli algoritmi quantistici: si tratta di tecniche non basate sulla fattorizzazione nè sul logaritmo discreto Chen et al. [2016].
In questo capitolo, per iniziare, faremo una panormanica del computer quantistico, spiegando più nel dettaglio cosa si intende per sovrapposizione di stati (sezione 2.2). Nella sezione 2.3 diremo passo dopo passo come funzione l'algoritmo di Shor e ne presenteremo un piccolo esempio per capire come opera. Nell'ultima sezione, infine, parleremo del rapporto con la crittografia classica, in particolare con RSA, per vedere se sia effettivamente in pericolo e quali contromisure si stanno adottando.
Chiameremo *computer classico* (o semplicemente computer) il computer non quantistico.
Tranne dove diversamente specificato, il materiale di questo capitolo è tratto da Hayward [originale 1999, modificate nel 2015].

2.2 Quantum Computing

I computer sono basati sui *bit*. Un bit rappresenta la più piccola unità di informazione, e può valere 0 o 1. Fisicamente, i bit sono memorizzati su diversi supporti fisici, come possono ad esempio essere i circuiti elettrici o i chip al silicone. Possiamo concettualmente immaginare che il più piccolo supporto fisico per il bit sia una particella subatomica (protone, elettrone o neutrone) avente spin $+1/2$ (bit 1) o $-1/2$ (bit 0) . Questa ipotesi si rivela però essere sbagliata: stiamo infatti applicando dei principi della fisica classica in un campo, quello subatomico, in cui non sono validi. Diciamo perciò che una particella subatomica con spin $\pm 1/2$ è una delle possibili

rappresentazioni fisiche del bit quantistico o *qubit*, che è l'unità fondamentale del *computer quantistico*, in maniera analoga al bit per il computer classico. Il computer quantistico funziona in base alle regole della fisica quantistica, di cui daremo un breve accenno.

Un qubit si comporta in modo analogo a un bit quando viene misurato: in questo caso lo spin può assumere o il valore $+1/2$ o $-1/2$. Tali valori (o stati) sono detti *autostati*. Diciamo in questo caso che il qubit *collassa*. Quando però non è misurato, può essere descritto come un vettore (che chiamiamo *vettore di stato*) in uno spazio di Hilbert. Tale vettore è una combinazione lineare a coefficienti complessi degli auotostati e si rappresenta usando la notazione di Dirac con il cosiddetto *ket vector*: $|\psi\rangle$, dove ψ è un elenco di numeri che contengono informazioni sulla proiezione del vettore di stato sugli autostati. Nella notazione di Dirac esiste anche il *bra vector*, che è un vettore riga, indicato con $\langle y|$; il prodotto tra vettore riga e vettore colonna è indicato con $\langle y|\psi\rangle$.
Come abbiamo già accennato, nella fisica quantistica vale il **principio di sovrapposizione**: un sistema può esistere in un mix di tutti i suoi possibili stati contemporaneamente.

Esempio 15. *Consideriamo sempre il caso di un sistema quantistico formato da due autostati e chiamiamo x_0 l'autostato corrispondente a spin $-1/2$ e x_1 quello con spin $+1/2$. Il vettore di stato $|X\rangle$ è dato in questo caso da:*

$$|X\rangle = w_0|x_0\rangle + w_1|x_1\rangle$$

dove w_0 e w_1 sono due numeri complessi detti pesi, che rappresentano il contributo di ciascun autostato al vettore di stato totale. Un vettore di stato si può anche rappresentare riportando solo i due pesi:

$$|X\rangle = (w_0, w_1)$$

Quando viene misurato, il qubit si trova in uno dei due autostati; il valore che assume in quel caso si indica quindi con

$$w_0|x_0\rangle + 0|x_1\rangle = (w_0,0)$$

se il vettore si trova in x_0, o con

$$0|x_0\rangle + w_1|x_1\rangle = (0, w_1)$$

se il vettore si trova in x_1.

	Computer classico	**Computer quantistico**
Unità di memoria	bit	qubit
Segue le regole di	fisica classica	fisica quantistica
Può esistere in	uno stato per volta	una sovrapposizione di stati

Tabella 2.1: Differenze tra computer classico e computer quantistico

2.2.1 Sistema quantistico a n stati

Fino ad ora abbiamo parlato di sistema quantistico a due stati, ma il concetto si può facilmente generalizzare a sistemi con n stati. In questo caso, gli autostati sono $x_0, x_1, \ldots, x_{n-1}$, mentre un vettore di stato è dato da:

$$|X\rangle = w_0|x_0\rangle + w_1|x_1\rangle + \cdots + w_{n-1}|x_{n-1}\rangle = \sum_{i=0}^{n-1} w_i|x_i\rangle \tag{2.1}$$

e lo possiamo indicare anche usando solo i valori dei pesi:

$$|X\rangle = \begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_{n-1} \end{pmatrix} \tag{2.2}$$

Possiamo servirci di queste informazioni per costruire un *registro quantistico di memoria*, che è l'analogo quantistico del registro dei computer classici. Se il registro è formato da n qubit, il suo stato è descritto da una formula come la (2.1) o come la (2.2).

Osservazione 3. *Un registro classico di memoria con n bit può rappresentare 2^n valori, ma uno per volta; un registro quantistico, invece, in virtù del principio di sovrapposizione, è in grado di rappresentare tutti i 2^n valori contemporaneamente Quantum register - Wikipedia e può, per questo motivo, memorizzare un numero di informazioni che è esponenziale rispetto alla capacità del registro. Questo ci fa già intravedere come il computer quantistico risulti più efficiente del computer classico.*

2.3 Algoritmo di Shor

Al computer quantistico non viene data, fino all'inizio degli anni novanta, una grande importanza. Non se ne vede un immediato ritorno economico, e per questo il maggior ambito di ricerca rimane quello accademico. Tutto cambia nel 1994, quando Peter Shor, un matematico dei Bell Labs, scopre un algortimo quantistico che può fattorizzare un intero in un tempo polinomiale.

2.3.1 La funzione dell'algoritmo di Shor

L'algoritmo di Shor è basato sulla funzione $f(a) = x^a \pmod n$, dove:

- n è il numero da fattorizzare
- x è un intero coprimo ad n

Sappiamo che la funzione è periodica. Sia r il suo periodo. Poichè $x^0 = 1 \pmod n$, allora

$$x^r = x^{0+r} = x^0 \pmod n = 1$$
$$x^{2r} = x^r \cdot x^r = 1 \pmod n$$

e così via. Facendo alcuni passaggi algebrici, si ricava

$$x^r = 1 \pmod n$$
$$(x^{r/2})^2 = x^r = 1 \pmod n$$
$$(x^{r/2})^2 - 1 = 0 \pmod n$$

e se r è pari

$$(x^{r/2} + 1)(x^{r/2} - 1) = 0 \pmod n$$

Quindi, se $x^{r/2} \neq \pm 1$, il prodotto tra $(x^{r/2}+1)$ e $(x^{r/2}-1)$ è un multiplo intero di n. E questo significa che almeno uno dei due termini ha un fattore non banale in comune con n, che è possibile ottenere calcolando

$$mcd(x^{r/2}+1, n)$$

o

$$mcd(x^{r/2}-1, n)$$

2.3.2 Passaggi dell'algoritmo

I passaggi seguiti dall'algoritmo di Shor per trovare il periodo r di $f(a)$ sono:

1. Stabilire se il numero n da fattorizzare è o un numero primo o l'elevamento a potenza di un numero primo. In caso di risposta affermativa, esistono degli algoritmi efficienti per computer classici in grado di fattorizzarlo.

2. Scegliere un intero q potenza di 2 e tale che $n^2 \leq q \leq 2n^2$.

3. Scegliere un numero casuale x coprimo di n.

4. Creare un registro quantistico di memoria e suddividerlo in due parti; in questo modo il suo stato si può descrivere con il ket vector $|reg1, reg2\rangle$. Il primo registro deve contenere abbasatanza qubit per rappresentare tutti gli interi da 0 a $q-1$, mentre il secondo deve poter rappresentare gli interi tra 0 e $n-1$.

5. Nel registro 1 caricare una sovrapposizione equipesata di tutti gli stati tra 0 e $q-1$; il secondo registro, invece, è inizializzato con tutti 0. Il vettore di stato è:

$$\frac{1}{\sqrt{q}} \sum_{a=0}^{q-1} |a,0\rangle$$

6. Nel registro 2 caricare il risultato della trasformazione $x^a \pmod n$ per ogni valore di a presente nella sovrapposizione del registro 1. Come abbiamo detto nell'osservazione 3, il computer quantistico può fare questa operazione in un solo passaggio. Il vettore di stato diventa quindi:

$$\frac{1}{\sqrt{q}} \sum_{a=0}^{q-1} |a, x^a \bmod n\rangle$$

7. Misurare il secondo registro e ottenere un valore k, facendo così collassare il primo regitro in un'equa sovrapposizione di tutti gli stati tra 0 e $q-1$ tali che $x^a \bmod n = k$. Lo stato del registro dopo questo passaggio è:

$$\frac{1}{\sqrt{|A|}} \sum_{a' \in A} |a', k\rangle$$

   dove A è l'insieme di tutti gli a' tali che $x^{a'} = k \pmod n$.

8. Calcolare la trasformata di Fourier discreta sul registro 1. In generale, la trasformata discreta di Fourier è data da :

$$\mathcal{F}(|a\rangle) = \frac{1}{\sqrt{q}} \sum_{q=0}^{q-1} |c\rangle \cdot e^{2\pi i ac/q}$$

Lo stato del registro diventa quindi:

$$\frac{1}{\sqrt{|A|}} \sum_{a' \in A} \frac{1}{\sqrt{q}} \sum_{c=0}^{q-1} |c, k\rangle \cdot e^{2\pi i a' c/q} \tag{2.3}$$

La trasformata di Fourier ha l'effeto di massimizzare la probabilità di trovare, nella prima parte del registro, un mutiplo intero m di q/r, dove r è il periodo della funzione cercata.

9. Misurare lo stato m del registro 1.

10. Processare il valore m, in modo da calcolare r sulla base di quanto si conosce di m e di q. Sappiamo infatti che
$$m = \lambda \cdot \frac{q}{r}$$
per qualche $\lambda \in \mathbb{Z}$; viene quindi eseguita la divisione in floating point tra m e q, che sappiamo essere uguale a $\frac{\lambda}{r}$. Si calcola poi la migliore frazione $\frac{x}{y}$ che approssima questo risultato, tenendo come vincolo $y \leq q$, dove y è il candidato ad essere r. Se y e dispari, lo si raddoppia, tenendo comunque presente che deve rimanere più piccolo di q.

11. Trovato r, procedere a calcolare i due mcd tra $(x^{r/2}+1, n)$ e $(x^{r/2}-1, n)$. Se si ottiene un fattore non banale di n, l'algoritmo si ferma; altrimenti occorre ripartire dal passo 4.

I primi tre passaggi e il calcolo dei qubit necessari per il registro quantistico di memoria sono eseguiti da un computer classico, i passaggi dal 4 al 9 da un computer quantistico e gli ultimi due passaggi nuovamente da un computer classico (v. tabella 2.2).

Computer Classico	**Computer Quantistico**
Verificare se n è primo o potenza di primo	
Scegliere q potenza di 2 e tale che $n^2 \leq q \leq 2n^2$	
Scegliere x coprimo di n	
Calcolare qubit per il registro di memoria	
	Scrivere i dati sul registro 1
	Scrivere nel registro 2 i risultati di $x^a \bmod n$
	Misurare k dal secondo registro
	Calcolare la trasformata discreta di Fourier sul registro 1
	Misurare lo stato m del registro 1
Post-processing di m	
Calcolare $\text{mcd}(x^{r/2} \pm 1, n)$	

Tabella 2.2: Algoritmo di Shor: passaggi eseguiti dal computer classico e dal computer quantistico

Osservazione 4. *L'ultimo step sottointende il fatto che l'algoritmo di Shor può anche fallire. Ad esempio, può accadere che la trasformata di Fourier del passaggio 8 dia come risultato* 0*, rendendo poi inutile il post-processing. O ancora, è possibile ottenere come risultato i due fattori banali* 1 *e* n*. Infatti, se si vuole fattorizzare* $n = 21$*, scegliendo* $x = 4$*, l'algoritmo può trovare* $r = 6$*. Calcolando i due massimi comuni divisori, si ottiene:*

$$\text{mcd}(4^3 + 1{,}21) = \text{mcd}(65{,}21) = 1$$
$$\text{mcd}(4^3 - 1{,}21) = \text{mcd}(63{,}21) = 21$$

Esempio 16. *Riportiamo l'esperimento fatto nel 2001 che ha consentito di fattorizzare* $n = 15$ *con l'algoritmo di Shor e che si trova descritto dettagliatamente in Vandersypen et al. [2001]. La funzione di cui dobbiamo trovare il periodo è:* $f(a) = x^a \pmod{15}$. *Prima di iniziare con l'algoritmo, facciamo alcuni passaggi che ci permettono di semplificare i conti.*
Sia $a_{N-1}a_{N-2}\dots a_1a_0$ *la rappresentazione binaria di* a, *dove* N *è il numero di bit. Allora l'elevamento a potenza* x^a *è dato da:*

$$x^a = x^{2^{N-1}a_{N-1}} x^{2^{N-2}a_{N-2}} \dots x^{2a_1} x^{a_0} = \prod_{k=0}^{N-1} x^{2^k a_k} \tag{2.4}$$

Sappiamo anche che x *deve essere coprimo a* 15. *Se prendiamo* $x = 11$, *si ha che* $11^2 = 1 \pmod{15}$, *da cui* $x^{2^k} = 1 \bmod 15\ \forall k \geq 1$. *La produttoria della formula (2.4) si semplifica:*

$$\prod_{k=0}^{N-1} x^{2^k a_k} = x^{a_0} \pmod{15}$$

Per questo motivo nella prima parte del registro quantistico di memoria possiamo usare anche solo un qubit. Si comportano in modo analogo anche i valori $x = 4$ *e* $x = 14$. *Invece se* $x = 2$, *si ha che* $2^4 \pmod{15} = 1$, *da cui segue che* $x^{2^k} = 1 \pmod{15}\ \forall k \geq 2$. *In questo caso, la formula (2.4) diventa quindi:*

$$\prod_{k=0}^{N-1} x^{2^k a_k} = x^{2a_1} x^{a_0}$$

e possiamo quindi usare due qubit per il registro 1. In modo analogo, si comportano anche i valori $x = 7$, $x = 8$ *e* $x = 13$.
Mettendo insieme i due casi, conviene tuttavia lavorare con un registro 1 composto da 3 *qubit, in quanto si ha la possibilità di trovare più periodi. La seconda parte del registro sarà invece formata, seguendo le regole classiche, da* 4 *qubit.*
L'esperimento prosegue poi scegliendo $x = 11$, *caricando il registro quantistico come abbiamo visto e calcolando la funzione* $11^a \bmod 15$. *Dall'analisi, il qubit 1 e il 2 risultano essere allo stato* $|0\rangle$, *mentre il terzo è in un equa sovrapposizione degli stati* $|0\rangle$ *e* $|1\rangle$. *Dopo aver calcolato la trasformata inversa di Fourier, il qubit più significativo è il terzo: il registro si trova quindi in una sovrapposizione di* $|000\rangle$ *e* $100\rangle$ *o, in decimale,* $|0\rangle$ *e* $|4\rangle$. *Si ha quindi* $m = 4$ *e inoltre, poichè* $q = 2^N = 8$, *allora* $r = \frac{8}{4} = 2$. *Infine si ha:*

$$\text{mcd}(11^{2/2} + 1{,}15) = \text{mcd}(12{,}15) = 3$$
$$\text{mcd}(11^{2/2} - 1{,}15) = \text{mcd}(10{,}15) = 5$$

dove 3 *e* 5 *sono proprio due fattori di* 15 *non banali.*

2.4 Effetti su RSA

La scoperta di Shor non determina, tuttavia, l'abbandono di RSA, che ancora oggi è uno dei sistemi crittografici più usati. Lo sviluppo tecnologico del computer quantistico si imbatte infatti in alcuni ostacoli difficili da superare. Tra questi occore ricordare il fenomeno della *decoerenza*, cioè la perdita delle sue caratteristiche quantistiche quando messo a contatto con un ambiente esterno Quantum Decoherence - Wikipedia.
A questo punto però è lecito domandarsi se questi ostacoli verrano a breve superati e se l'arrivo

della tecnologia quantistica sul mercato sia imminente. Per (provare a) rispondere a questa domanda, analizziamo brevemente la storia del computer quantistico, dall'algoritmo di Shor fino ai giorni nostri.

2.4.1 Storia del computer quantistico

Questa sezione è tratta da Quantum Computing- Wikipedia e da altre fonti che verranno citate man mano.
Come abbiamo già accennato, lo sviluppo del computer quantistico conosce un'accelerazione dopo la pubblicazione dell'algoritmo di Shor nel 1994. Nel 1996 viene pubblicato l'articolo di DiVincenzo sui requisiti che un computer quantistico deve soddisfare DiVincenzo [1996], ma è solo cinque anni più tardi, nel 2001, che l'algoritmo di Shor è usato per la prima volta per fattorizzare il numero 15 Vandersypen et al. [2001]. Dobbiamo però aspettare il 2011 perchè interi più grandi, come 21 o addirittura come 143, vengano fattorizzati da alcune rudimentali inplementazione di un computer quantistico (Martin-Lopez et al. [2011] e Xu et al. [2011]). Un anno dopo, un gruppo di ricerca guidato da alcuni ingegneri australiani riesce a creare il primo qubit funzionante, basato su un atomo di silicone First quantum bit. Nel 2013, Google comunica la nascita del *Quantum Artificial Intelligence Lab*, un laboratorio di ricerca con l'obiettivo di studiare il computer quantistico nell'ambito del machine learning Neven [2013]. Due anni più tardi, il laboratorio pubblica un paper in cui afferma di essere riuscito a far funzionare un algoritmo quantistico sulla propria macchina Hardesty [2015]. Nel 2016, IBM annuncia il primo servizio cloud di quantum computing IBM cloud computing. Nel 2017, è sempre IBM che crea il primo sistema a 56 qubit Staff [2017], che viene però superato dal processore testing di Google a 72 qubit l'anno seguente Curtis [2018]. A inizio 2019, l'IBM crea il primo computer quantistico da usare al di fuori dal laboratorio Lardinois [2019], che però sembra essere ancora lontano dal diventare disponibile a livello commerciale.

Anno	**Avvenimento**
1994	Algoritmo di Shor per la fattorizzazione
1996	Requisiti del computer quantistico (DiVincenzo)
2001	Fattorizzazione di 15 con l'algoritmo di Shor
2011	Fattorizzazione di 21 e di 143
2012	Primo qubit funzionante
2013	Quantum Artificial Intelligence Lab (Google)
2016	Primo servzio cloud per quantum computing (IBM)
2017	Primo sistema a 56 qubit (IBM)
2018	Processore testing a 72 qubit (Google)
2019	Primo computer quantistico da usare fuori dal laboratorio (IBM)

Tabella 2.3: Storia del computer quantistico, eventi principali

Da questa breve panoramica emerge come RSA sembri quindi essere ancora sicuro. Ma per quanto lo sarà ancora? Mentre i team di ricerca delle aziende tecnologiche lavorano per mettere a punto il computer quantistico, chi si occupa di crittografia studia nuove tecniche che non necessitano più della fattorizzazione. Tipici esempi sono quelli basati sui codici correttori, di cui il capostipite è il sitema di McEliece.

Parte II

Crittografia post-quantum: sistema di McEliece e codice di Fibonacci

Capitolo 3

Sistema Crittografico di McEliece

3.1 Introduzione

Come costuire sistemi crittografici resistenti agli attacchi del computer quantistico? La ricerca in questo ambito si dirama in due direzioni, a seconda che gli algoritmi siano fatti per computer quantistici o per computer classici Chen et al. [2016]. In questo secondo ambito, che è definito *crittografia post-quantum*, una delle strade seguite è quella dei *codici correttori*. Questa espressione indica un insieme di procedimenti usati quando si mandano i messaggi per individuare e eventualmente correggere gli errori di trasmissione Cotignoli [2018].
Nel 1978, Robert McEliece sviluppa quello che può essere considerato il primo sistema crittografico basato sui codici correttori Perkins [2019]. Questo sistema sfrutta le caratterstiche, in particolare, del codice di Goppa; è facile e intuitivo da comprendere, e al contrario è difficile da rompere. Il suo svantaggio più grosso consiste nel fatto che non può essere usato per gli schemi di firma digitale.
La parte centrale di questo capitolo è la sezione 3.5, in cui analizzeremo proprio il sistema di McEliece, presentandone il funzionamento e vedendone i vataggi e gli svantaggi. Le sezioni precedenti forniscono una base, sicuramente non esaustiva, per comprenderlo dal punto di vista matematico: daremo prima delle informazioni aggiuntive sui campi finiti (sezione 3.2), per poi fare una panoramica generale sui codici correttori (sezione 3.3), soffermandoci in particolare sul codice di Goppa.

3.2 Campi finiti

Indichiamo con $\mathbb{F}_q$ un campo finito avente q elementi. Ricordiamo che, come abbiamo visto nel primo capitolo, un **campo** è un anello R per cui sia $(R, +)$ sia $(R, -)$ sono gruppi abeliani.
Diamo ora la definizione di ordine e caratteristica di un campo, che prendiamo da Huffman and Pless [2003].

Definizione 6. *L'ordine di un campo finito $\mathbb{F}_q$ è il numero degli elementi al suo interno. Ad esempio il campo $\mathbb{Z}_p$, dove p è un numero primo, ha ordine p.*
Sia invece $x \in \mathbb{F}_q$. Si definisce ordine di x (rispetto all'addizione) il più piccolo intero $z > 0$ per cui si ha

$$\underbrace{x + x + x + \cdots + x}_{z-\text{volte}} = x \cdot n = 0$$

Con la notazione che abbiamo introdotto, segue immediatamente che l'ordine di un campo finito $\mathbb{F}_q$ è proprio q.
La duplice definizione di ordine vale anche in maniera analoga per i gruppi finiti Group - Wikipedia:

- l'ordine di un gruppo finito G, indicato con $|G|$, è il numero di elementi di G
- l'ordine di un elemento $x \in G$ è il più piccolo intero z per cui si ha

$$\underbrace{x + x + x + \cdots + x}_{\text{z}-\text{volte}} = x \cdot n = 0$$

Definizione 7. *La caratteristica di un campo finito è l'ordine dell'identità moltiplicativa 1, ossia il più piccolo intero m per cui si ha:*

$$\underbrace{1 + 1 + 1 + \cdots + 1}_{\text{m}-\text{volte}} = 1 \cdot m = 0$$

La caratteristica deve soddisfare la seguente proprietà:

Proprietà 6. *La caratteristica di un campo è sempre o zero o un numero primo*

Dimostrazione. Riprendiamo la dimostrazione da characteristic of a field e da characteristic of a field is a prime.
Sia m la caratteristica di un campo finito $\mathbb{F}_q$, con $m \neq 0$. Facciamo vedere che:

1. se $m \neq 0$, allora m deve essere un numero primo;
2. se m non è un numero primo, allora deve essere $m = 0$

Per il caso 1, prendiamo $m \neq 0$ e dimostriamo che m deve essere primo. Per assurdo, sia m un numero non primo; in questo caso, si può scrivere $m = m_1 m_2$, con $0 < m_1, m_2 < m$. Da qui segue:

$$\underbrace{\underbrace{(1 + 1 + \cdots + 1)}_{\text{m}_1-\text{volte}} + \cdots + \underbrace{(1 + 1 + \cdots + 1)}_{\text{m}_1-\text{volte}}}_{\text{m}_2-\text{volte}} = 0$$

Definiamo $a = \underbrace{1 + 1 + \cdots + 1}_{\text{m}_1-\text{volte}}$, da cui $\underbrace{a + a + \ldots a}_{\text{m}_2-\text{volte}} = 0$. Inoltre, dalla proprietà di chiusura discussa nel capitolo 1, segue che $a \in \mathbb{F}_q$.
Ci sono due possibilità.

- Se $a = 0$, allora $\underbrace{1 + 1 + \cdots + 1}_{\text{m}_1} = 0$; questo vuol dire che a è la caratteristica del campo. Ma questo è assurdo, perchè $a < m$, ed m deve essere il più piccolo intero tale che $\underbrace{1 + 1 + \cdots + 1}_{m-volte}$ è nullo.
- Se $a \neq 0$, allora a ha un inverso moltiplicativo a^{-1} tale che $aa^{-1} = 1$. Si ha quindi:

$$0 = 0 \cdot a^{-1} = \underbrace{(a + a + \cdots + a)}_{\text{m}_2} a^{-1} = \underbrace{aa^{-1} + aa^{-1} + \cdots + aa^{-1}}_{\text{m}_2} = \underbrace{1 + 1 + \cdots + 1}_{\text{m}_2}$$

da cui m_2 è caratteristica del campo. Ma anche questo è assurdo, perchè $m_2 < m$.

Per il caso 2, poniamo che m non sia un numero primo, e che possiamo quindi scriverlo come $m = m_1 \cdot m_2$. Facciamo quindi vedere che deve per forza essere $m = 0$. Dalla definizione di caratteristica, abbiamo che $m \cdot 1 = 0$, da cui $m_1 m_2 \cdot 1 = 0$. Ma $1 \neq 0$, quindi, dalle proprietà di campo, deve per forza essere $m_1 m_2 = m = 0$. □

Introduciamo adesso il teorema di Cauchy e il suo corollario che ci serviranno per dimostrare la proprietà sull'ordine di un campo finito, che useremo per costruire i campi finiti che ci servono per i codici correttori. Il teorema è tratto da Cauchy's theorem - Wikipedia.

Teorema 7. *(di Cauchy) Sia G un gruppo finito di ordine $|G|$ e p un numero primo. Se p divide $|G|$, allora G ha un elemento di ordine p.*

La dimostrazione di questo teorema è presa da C.Pinter [2010] e viene fatta nel caso di gruppo additivo.

Dimostrazione. Consideriamo tutte le possibili tuple di lunghezza p del tipo $(a_1, a_2, \dots a_p)$, con $a_i \in G$ e tali che

$$a_1 + a_2 + \cdots + a_p = \sum_{i=1}^{p} a_i = 0 \tag{3.1}$$

dove 0 indica l'identità additiva del gruppo G.
Come si possono scegliere gli elementi di tali tuple? Possiamo prendere in modo casuale tutti gli elementi a_1, a_2, … fino ad a_{p-1}; l'ultimo elemento a_p deve essere

$$a_p = -a_1 - a_2 \cdots - a_{p-1}$$

in modo da soddisfare l'equazione (3.1) (la notazione $-a_i$ indica l'inverso additivo di a_i). Le tuple così definite risultano quindi essere in totale $|G|^p - 1$, dove l'ordine di G è il numero dei suoi elementi.
Due o più tuple di lunghezza p sono equivalenti se si possono ottenere l'una dall'altra tramite una permutazione ciclica; la tupla $(a_1, a_2, \dots, a_p)$ sarà quindi equivalente a $(a_2, a_3, \dots a_p, a_1)$, a $(a_3, a_4, \dots a_{p-1}, a_p, a_1, a_2)$, e così via fino a $(a_p, a_1, a_2, \dots, a_{p-1})$. Ciascuna tupla di lunghezza p forma perciò una classe di equivalenza avente esattamente p elementi (dove p in questo caso è il numero delle permutazioni circolari). Le uniche tuple che formano una classe di equivalenza con un solo membro sono quelle aventi lo stesso elemento ripetuto p volte. Ne esiste sicuramente una che soddisfa l'equazione (3.1): è quella formata da tutti 0. Ce ne sono altre? Per assurdo, se non ce ne fossero, significherebbe che il numero totale di classi di equivalenza con p elementi sarebbe $|G|^{p-1} - 1$, e questo significherebbe che p divide $(|G|^{p-1} - 1)$, cioè che

$$|G|^{p-1} = 1 \pmod{p}$$

Ma questo è assurdo, perchè p deve dividere $|G|$, da cui $|G|^{p-1} = 0 \pmod{p}$. Perciò devono esistere altre classi di equivalenza formate da un solo elemento ripetuto p volte:

$$\underbrace{(a, a, \dots, a)}_{p-volte}$$

dove a è tale che $\underbrace{a + a + \dots a}_{p-volte} = 0$, cioè a ha ordine p. □

Corollario 1. *Sia G un $p-$gruppo finito, ossia un gruppo in cui ogni elemento ha per ordine una potenza di p, con p numero primo (p-group - Wikipedia). Allora $|G| = p^k$ per qualche intero k.*

Dimostrazione. Sia G un $p-gruppo$ finito, con p un numero primo, avente ordine $|G|$. Per assurdo, sia $|G| \neq p^k$. Questo significa che c'è un altro primo p', $p' \neq p$, tale che p' divide $|G|$. Allora, per il teorema di Cauchy, esiste un elemento $x \in G$ tale che x ha ordine p'. Ma questo è assurdo, perchè G è un $p-gruppo$ e quindi x deve avere ordine p^k per qualche intero k. Dunque deve per forza essere $|G| = p^k$. □

Osservazione 5. *Nella fonte citata, il corollario 1 è valido anche nella direzione opposta: se un gruppo finito G ha ordine p^k, allora G è un $p-gruppo$ finito. Questo enunciato e la sua dimostrazione, che implica il teorema di Lagrange, sono omessi perchè esulano dai contenuti trattati.*

Abbiamo visto che l'insieme $\mathbb{Z}_n$ degli interi in modulo n è un campo finito se e solo se n è un numero primo. Indichiamo questi campi con $\mathbb{F}_p$. Parlando di codici correttori occorre però introdurre i campi $\mathbb{F}_q$ o $GF(q)$, dove q indica l'ordine del campo e deve soddisfare la seguente proprietà:

Proprietà 7. *L'ordine q di un campo è un primo p o una potenza di un primo p; in sintesi, $q = p^k$, con $k \geq 1$.*

Facciamo vedere la dimostrazione di questa proprietà, che prendiamo da Order of finite fields.

Dimostrazione. Sia p la caratteristica del campo finito $\mathbb{F}_q$. Consideriamo il gruppo additivo $(\mathbb{F}_q, +)$. Sappiamo che p è un numero primo. Sia $x \in \mathbb{F}_q$, $x \neq 0$. Allora, per la proprietà distributiva del campo:

$$x \cdot p = x \cdot (1 \cdot p) = x \cdot 0 = 0$$

Inoltre, per la definizione di caratterstica, p è il più piccolo intero per cui $1 \cdot p = 0$, che implica che x ha ordine p. Ma questo implica che $(\mathbb{F}_q, +)$ è un $p-gruppo$. Dal corollario 1, segue che $\mathbb{F}_q$ ha ordine p^k. □

Nella prossima sezione vedremo come costruire dei campi finiti di cardinalità 2^n e faremo un esempio di campo con $n = 4$.

3.2.1 Costruzione di campi finiti

Questa sezione e quella successiva sono tratte da McAndrew [2012] e da finite fields - Wikipedia. In generale, i campi finiti $GF(p^n)$ vengono generati come $GF(p)/(f(x))$, dove $f(x)$ è un polinomio irriducibile di grado n a coefficienti in $GF(p)$. In questo paragrafo, vediamo in dettaglio come ottenere campi finiti del tipo $GF(2^n)$, che sono poi quelli che useremo per il codice di Goppa nella sezione 3.4.
Per generare i campi finti di ordine 2^n occorre seguire questi passaggi:

1. come elementi del campo, prendiamo tutti i polinomi di grado $n-1$ e aventi coefficienti in $\mathbb{Z}_2$; questi polinomi si possono vedere anche come stringhe binarie di lunghezza n, in cui ogni bit rappresenta i coefficienti; a loro volta, le stringhe si possono vedere come la rappresentazione binaria dei numeri tra 0 e $n-1$;

2. l'operazione di addizione è l'addizione polinomiale in modulo 2; considerando le stringhe binarie, l'addizione corrisponde allo XOR bit a bit;

3. l'operazione di moltiplicazione è fatta in modulo $f(x)$, dove $f(x)$ è un polinomio irriducibile di grado n; non importa quale polinomio scegliamo, il campo che ne risulta è lo stesso a meno di isomorfismi.

Esempio 17. *Costruiamo il campo $GF(2^4)$, o $\mathbb{F}_{16}$, scegliendo come polinomio irriduciblie $f(x) = x^4 + x + 1$. I polinomi e la loro rappresentazione binaria sono riportati nella tabella 3.1*

rappresentazione binaria	polinomio
0000	0
1000	1
0100	x
1100	$1 + x$
0010	x^2
1010	$1 + x^2$
0110	$x + x^2$
1110	$1 + x + x^2$
0001	x^3
1001	$1 + x^3$
0101	$x + x^3$
1101	$1 + x + x^3$
0011	$x^2 + x^3$
1011	$1 + x^2 + x^3$
0111	$x + x^2 + x^3$
1111	$1 + x + x^2 + x^3$

Tabella 3.1: $GF(2^4)$

3.2.2 Moltiplicazione in un campo finito

Abbiamo visto nella sezione 3.2.1 che la moltipicazione nei campi finiti si comporta come la moltiplicazione tra polinomi modulo $f(x)$. Esiste tuttavia un modo più semplice di operare, che sfrutta il concetto di generatore.
Supponiamo quindi di avere a disposizione $g(x)$, generatore di $\mathbb{F}_q$, e di dover eseguire la moltiplicazione tra $a(x)$ e $b(x)$, con $a(x), b(x) \in \mathbb{F}_q$

- scriviamo i due polinomi $a(x)$ e $b(x)$ in funzione di $g(x)$:

$$a(x)b(x) = g(x)^k g(x)^l = g(x)^{k+l}$$

- semplifichiamo l'esponente modulo $q - 1$: $g(x)^{k+l} = g(x)^h$, con $h = k + l \pmod{(q-1)}$; questo si può fare perchè $g(x)^{q-1} = 1$ in $\mathbb{F}_q$

- calcoliamo il valore della potenza $g(x)^h$

Esempio 18. *Riprendiamo il campo $\mathbb{F}_{16}$ visto nell'esempio 17. Un generatore per tale campo è $g(x) = x$. Infatti:*

$$
\begin{aligned}
x^0 &= 1 \\
x^1 &= x \\
x^2 &= x^2 \\
x^3 &= x^3 \\
x^4 &= 1 + x \\
x^5 &= x + x^2 \\
x^6 &= x^2 + x^3 \\
x^7 &= 1 + x + x^3 \\
x^8 &= 1 + x^2 \\
x^9 &= x + x^3 \\
x^{10} &= 1 + x + x^2 \\
x^{11} &= x + x^2 + x^3 \\
x^{12} &= 1 + x + x^2 + x^3 \\
x^{13} &= 1 + x^2 + x^3 \\
x^{14} &= 1 + x^3 \\
x^{15} &= 1
\end{aligned}
$$

La moltiplicazione, ad esempio tra $x + x^3$ *e* $1 + x + x^2$ *diventa quindi quella tra* x^9 *e* x^{10}*:*

$$(x + x^3)(1 + x + x^2) = x^9 x^{10} = x^{19} = x^4 = 1 + x$$

dove per passare da x^{19} *a* x^4 *abbiamo calcolato* $19 \bmod 15 = 4$*.*

3.3 Codici correttori

In questa sezione usiamo il materiale di Huffman and Pless [2003], code theory, Sevalnev [2008] e Cotignoli [2018].
I codici correttori sono dei procedimenti per individuare e, quando possibile, correggere gli errori contenuti in un messaggio, dove il messaggio è una sequenza di numeri. Vediamo alcuni esempi introduttivi per poi successivamente dare la definizione matematica di codice.

Esempio 19. *Supponiamo di avere a disposizione un codice binario, cioè in cui si possono usare solo i simboli* 0 *e* 1*; in questo caso diciamo che l'alfabeto del codice è l'insieme* {0,1}*. Decidiamo di codificare le lettere dalla A alla H con 3 bit ciascuna:*

- *A=000*
- *B=001*
- *C=010*
- *D=011*

- *E=100*
- *F=101*
- *G=110*
- *H=111*

Ipotizziamo che il mittente voglia spedire la parola inglese DAD, che viene quindi codificata come 011 000 011. *Se non c'è nessun errore, il destinatario riceve la stringa corretta e decodifica la parola che gli era stata inviata. Cosa succede invece nel caso di errori? Si osserva immediatamente che in questo caso basta anche solo sbagiare un bit perchè il destinatario riceva la parola sbagliata. Per esempio, al posto della stringa corretta per DAD può ricevere la stringa* 001 00 011, *che corrisponde alla parola BAD. Con questo tipo di codice, quindi, non è possibile nè individuare nè correggere gli errori.*

Ovviamente esistono delle tecniche per ovviare a questo problema. Negli esempi 20 e 21 vedremo come è possibile, date alcune premesse, indviduare e correggere gli errori

Esempio 20. *Codifichiamo di nuovo le lettere dalla A alla H, ma stavolta usiamo 4 bit per ogni lettera invece di 3, e facciamo in modo che ogni lettera abbia un numero pari di bit a 1 (facciamo quella che viene chiamata binary check parity):*

- *A=0000*
- *B=0011*
- *C=0101*
- *D=0110*
- *E=1001*
- *F=1010*
- *G=1100*
- *H=1111*

Anche in questo caso, spediamo il messaggio DAD. Osserviamo che il destinatario, nel caso di un errore, può solo trovarlo ma non correggerlo: se infatti la stringa che riceve è 0110 0001 0110, nota subito che l'errore è nel secondo blocco, che contiene un numero dispari di bit a 1; non può però correggerlo, in quanto ci sono più lettere "a distanza minima" da 0001: la A, la B, la C e la E; daremo poi una definizione formale di cosa intendiamo per "distanza minima" nella sezione 3.3.1, per ora accenniamo solo che le possibili lettere sono quelle con un solo bit diverso dalla stringa ricevuta.

Esempio 21. *In questo esempio le solite lettere vengono codificate con 9 bit:*

- *A=000 000 000*
- *B=001 001 001*
- *C=010 010 010*
- *D=011 011 011*

- *E=100 100 100*
- *F=101 101 101*
- *G=110 110 110*
- *H=111 111 111*

In questo caso, il mittente spedisce la parola DAD, che viene quindi codificata come 011 011 011 000 000 000 011 011 011. Nel caso i cui venga commesso un solo errore, è possibile non solo individuarlo ma anche correggerlo: non ci sono dubbi su qual è la lettera a "distanza minima". Infatti, se il messaggio che arriva è 011 011 011 000 010 000 011 011 011, il destinatario, dopo averlo diviso in blocchi da 9 bit, individua l'errore (un bit nel secondo blocco) e può correggerlo, sostituendo 000 010 000 con 000 000 000, che corrisponde alla lettera A. In questo caso, tuttavia, abbiamo usato un numero eccessivo di bit per poi correggerne uno solo.

Questi esempi mettono in luce come l'individuazione e correzione di errori sia possibile grazie ad alcuni bit extra, che non servono per codificare.
Diamo adesso una definizione più formale di codice e di codice lineare. Sia $\mathbb{F}_q^n$ lo spazio vettoriale delle tuple di lunghezza n definite sul campo $\mathbb{F}_q$, dove il fatto che $\mathbb{F}_q^n$ sia uno spazio vettoriale segue dalle proprietà elencate nel capitolo 1.

Definizione 8. *Un (n, M) codice $\mathcal{C}$ su $\mathbb{F}_q$ è definito come un sottoinsieme di $\mathbb{F}_q^n$ di dimensione M.*

I codici su $\mathbb{F}_2$ sono detti *codici binari*. Un vettore del tipo $(c_1, \ldots, c_n)$ è chiamato *parola del codice* o *codeword*, e si può indicare anche con $c_1 \ldots c_n$.
Un importante tipo di codici sono i codici lineari.

Definizione 9. *Un codice $\mathcal{C}$ è detto lineare se forma un sottospazio di dimensione k dello spazio vettoriale $\mathbb{F}_q^n$. Tali codici sono anche indicati con $\mathcal{C}[n, k]$.*

Per un codice lineare definito su $\mathbb{F}_q^n$, il numero di parole è q^k. Prima di continuare nel dettaglio a descrivere i codici (correttori), diamo una spiegazione schematica di come viene visto un canale di comunicazione:

- una sorgente S genera il messaggio, visto come un vettore $\mathbf{m}$ di lunghezza k: $\mathbf{m} = m_1 \ldots m_k$
- tramite un codificatore o encoder E, il messaggio è convertito in una parola del codice $\mathbf{c}$ di lunghezza n: $\mathbf{c} = c_1 \ldots c_n$
- la parola del codice è trasmessa all'interno di un canale di trasmissione C, generalmente disturbato da un qualche tipo di rumore che genera un errore $\mathbf{e} = e_1 \ldots e_n$
- il decodificatore o decoder D, che si trova all'altro capo del canale, riceve il vettore $\mathbf{y} = \mathbf{c}+\mathbf{e}$ ed ha il compito di calcolare $\hat{\mathbf{m}}$, una ricostruzione di $\mathbf{m}$

Quello che si vuole è che $\hat{\mathbf{m}} = \mathbf{m}$, ma questo non sempre accade. Questa spiegazione schematica del canale di counicazione è illustrata nella figura 3.1.

A livello intuitivo, la lunghezza n delle parole del codice è più grande di k, la lunghezza del messaggio, per generare quella che viene chiamata *ridondanza*, ossia l'aggiunta nel codice di elementi non presenti nel messaggio originale. La ridondanza serve a individuare e a correggere gli errori.

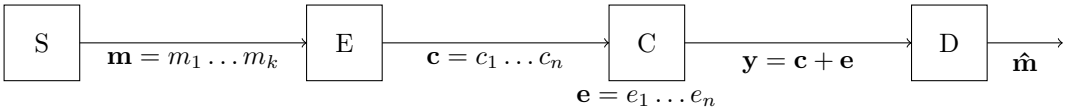

Figura 3.1: Illustrazione schematica del canale di comunicazione

3.3.1 Distanza e peso di un codice

Siano $\mathbf{c_1}, \mathbf{c_2} \in \mathbb{F}_q^n$ due parole del codice.

Definizione 10. *Si definisce distanza tra* $\mathbf{c_1}$ *e* $\mathbf{c_2}$ *il numero di componenti in cui* $\mathbf{c_1} \neq \mathbf{c_2}$ *e si indica con* $d(\mathbf{c_1}, \mathbf{c_2})$.

Esempio 22. *In* $\mathbb{F}_2^5$, $d(00111,11001) = 4$.

$\forall \mathbf{c_1}, \mathbf{c_2}, \mathbf{c_3} \in \mathbb{F}_q^n$, la distanza soddisfa queste proprietà:

1. $d(\mathbf{c_1}, \mathbf{c_2}) \geq 0$ e $d(\mathbf{c_1}, \mathbf{c_2}) = 0$ se e solo se $\mathbf{c_1} = \mathbf{c_2}$
2. $d(\mathbf{c_1}, \mathbf{c_2}) = d(\mathbf{c_2}, \mathbf{c_1})$
3. $d(\mathbf{c_1}, \mathbf{c_2}) \leq d(\mathbf{c_1}, \mathbf{c_3}) + d(\mathbf{c_2}, \mathbf{c_3})$

Definizione 11. *La distanza minima* $d(\mathcal{C})$ *di un codice* $\mathcal{C}$ *è la più piccola distanza tra due diverse parole del codice.*

$$d(\mathcal{C}) = \min\{d(\mathbf{c}_1, \mathbf{c}_2) | \mathbf{c}_1, \mathbf{c}_2 \in \mathcal{C}, \mathbf{c}_1 \neq \mathbf{c}_2\}$$

Come vedremo più avati, la distanza minima determina il numero di errori che è possibile individuare e correggere in una parola di codice.

Definizione 12. *Si definisce peso di una parola del codice* $\mathbf{c}$*, indicato con* $w(\mathbf{c})$*, il numero di componenti di* $\mathbf{c}$ *non nulle. Il peso di* $\mathbf{c}$ *si può anche vedere come* $d(\mathbf{c}, \mathbf{0})$.
In analogia con la distanza minima, si definisce peso minimo di un codice $\mathcal{C}$*, indicato con* $w(\mathcal{C})$*, il più piccolo peso di una parola del codice* $\mathbf{c}$ *non nulla:*

$$w(\mathcal{C}) = \min\{w(\mathbf{c}) | \mathbf{c} \in \mathcal{C}, \mathbf{c} \neq 0\}$$

Esiste un legame tra distanza e peso, come dimostra il seguente teorema.

Teorema 8. *Siano* $\mathbf{c}_1, \mathbf{c}_2 \in \mathcal{C}$ *due parole del codice. Allora* $d(\mathbf{c}_1, \mathbf{c}_2) = w(\mathbf{c}_1 - \mathbf{c}_2)$

Dimostrazione. La dimostrazione è immediata: è sufficiente osservare che il vettore $\mathbf{c}_1 - \mathbf{c}_2$ è non nullo in corrispondenza delle componenti i tali per cui $c_{1_i} \neq c_{2_i}$. □

Inoltre, per i codici lineari c'è una corrispondenza tra distanza minima e peso minimo.

Teorema 9. *Sia* $d(\mathcal{C})$ *la distanza minima di un codice lineare* $\mathcal{C}$*. Allora* $d(\mathcal{C}) = w(\mathcal{C})$.

Dimostrazione. Siano $\mathbf{c}_1, \mathbf{c}_2$ due parole del codice tali che $d(\mathbf{c}_1, \mathbf{c}_2) = d(\mathcal{C})$. Allora usando il teorema 8 abbiamo che:

$$d(\mathcal{C}) = d(\mathbf{c}_1, \mathbf{c}_2) = w(\mathbf{c}_1 - \mathbf{c}_2) \geq w(\mathcal{C})$$

perchè $\mathbf{c}_1, \mathbf{c}_2 \in \mathcal{C}$. Ma si ha anche che

$$w(\mathcal{C}) = w(\mathbf{z}) = d(\mathbf{z}, \mathbf{0}) \geq d(\mathcal{C})$$

□

Distanza minima e correzione di errori

Teorema 10. *Sia $\mathcal{C}$ un codice con distanza minima $d = d(\mathcal{C})$. Allora, per ogni parola del codice:*

- *è possibile individuare fino a s errori se $d \geq s + 1$*
- *è possibile correggere fino a r errori se $d \geq 2r + 1$*

Dimostrazione. Sia $d \geq s + 1$. Se viene trasmessa una parola di codice $\mathbf{c}$ e vengono commessi al più s errori, allora il vettore ricevuto non è una parola di codice e gli errori possono essere individuati.
Sia ora $d \geq 2r + 1$. Supponiamo che venga trasmessa la parola di codice $\mathbf{c}$ e venga ricevuto $\mathbf{y}$ contenente al più r errori. Allora $d(\mathbf{c}, \mathbf{y}) \leq r$. Sia $\mathbf{z}$ un'altra parola di codice tale che $d(\mathbf{z}, \mathbf{y}) \leq r$. Allora, usando le proprietà della distanza:

$$d(\mathbf{c}, \mathbf{z}) \leq d(\mathbf{c}, \mathbf{y}) + d(\mathbf{y}, \mathbf{z}) \leq 2r < d$$

per cui deve essere per forza $\mathbf{z} = \mathbf{c}$; in altre parole, $\mathbf{c}$ è la parola di codice più vicina a $\mathbf{y}$. □

Corollario 2. *Sia $\mathcal{C}$ un codice con distanza minima d. Allora è possibile individuare fino a $d-1$ errori e correggerne fino a $\lfloor \frac{d-1}{2} \rfloor$.*

Dimostrazione. Sia s il numero massimo di errori che si possono individuare. Abbiamo visto che $d \geq s + 1$, da cui segue che $s \leq d - 1$.
Sia ora r il numero massimo di errori che si possono correggere. Sappiamo che $d \geq 2r + 1$, da cui segue che $r \leq \lfloor \frac{d-1}{2} \rfloor$. □

3.3.2 Matrice di controllo parità e matrice generatrice

Quando abbiamo a che fare con un codice lineare, è utile gestirlo attraverso delle trasformazioni lineari. Per fare questo ci serviamo, prima di tutto, della matrice di controllo parità, che si può definire come segue:

Definizione 13. *La matrice di controllo parità H di un codice lineare $\mathcal{C}[n, k]$ è una matrice di dimensione $(n - k) \times n$ per cui si ha*

$$H \cdot \mathbf{c}^T = 0 \tag{3.2}$$

per ogni parola del codice $\mathbf{c}$.

Tale matrice non è unica; una matrice di controllo parità H che soddisfa l'equazione (3.2) e che useremo spesso nei calcoli si esprime nella forma:

$$H = [A|I_{n-k}] \tag{3.3}$$

dove A è una qualche matrice di dimensione $(n - k) \times k$, mentre I_{n-k} è la matrice identità di dimensione $(n - k) \times (n - k)$.
Il ruolo primcipale della matrice di controllo parità è quello di verificare la correttezza delle parole del codice $\mathbf{c}$ ricevute. Mostriamo due esempi in cui è fatto vedere come si determina H e come la si può anche usare per trovare tutte le parole del codice.

Esempio 23. *Per ricavare la matrice di controllo parità del codice binario dell'esempio 20, in cui $n = 4$ e $k = 3$, usiamo la formula (3.2), dove H sarà del tipo $H = \left[\begin{array}{ccc|c} h_1 & h_2 & h_3 & 1 \end{array}\right]$.*

$$\left[\begin{array}{ccc|c} h_1 & h_2 & h_3 & 1 \end{array}\right] \cdot \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} = 0$$

da cui si ricava l'equazione

$$h_1c_1 + h_2c_2 + h_3c_3 + h_4 = 0$$

Una possibile soluzione è $h_1 = h_2 = h_3 = 1$ *(stiamo infatti facendo i conti modulo 2). Quindi la matrice di controllo parità cercata è:* $H = \left[\begin{array}{ccc|c} 1 & 1 & 1 & 1 \end{array}\right]$.

Esempio 24. *Consideriamo il caso in cui il codice sia definito su* $\mathbb{Z}_2$ *e abbia matrice di controllo parità definita da:*

$$H = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

Vediamo come usare H *per ricavare il codice* $\mathcal{C}$*. Innanzitutto abbiamo che* $n = 4$ *e* $n - k = 2$*, da cui* $k = 2$*; le parole cercate saranno quindi del tipo* $\mathbf{c} = c_1c_2c_3c_4$ *e il codice avrà in totale* $2^k = 4$ *parole. La formula* $H \cdot \mathbf{c}^T = 0$ *dà origine al sistema:*

$$\begin{aligned} c_1 + c_3 &= 0 \\ c_1 + c_2 + c_4 &= 0 \end{aligned}$$

da cui il codice $\mathcal{C}$ *risulta essere:*

$$\mathcal{C} = \{0000, 0101, 1011, 1110\}$$

Oltre a quella di controllo parità, l'altra matrice che si usa è la matrice generatrice, cioè quella matrice che serve per trasformare un qualsiasi messaggio $\mathbf{m}$ in una parola del codice $\mathbf{c}$.

Definizione 14. *La matrice generatrice* G *di un codice lineare* $\mathcal{C}[n, k]$ *è quella (non unica) di dimensione* $k \times n$ *le cui righe sono una base per* $\mathcal{C}$*.*

Dalla definizione segue immediatamente che, per ogni messaggio $\mathbf{m} = m_1 \dots m_k$, la corrispondente parola del codice $\mathbf{c} = c_1 \dots c_n$ si ottiene moltiplicando $\mathbf{m}$ per G:

$$(m_1 \dots m_k) \cdot G = \begin{pmatrix} c_1 & \dots & c_n \end{pmatrix} \tag{3.4}$$

Che relazione esiste tra queste due matrici? Per i nostri scopi è sufficiente dimostrare il seguente teorema:

Teorema 11. *Sia* $\mathcal{C}[n, k]$ *un codice lineare avente matrice di controllo parità* H *e matrice generatrice* G*. Allora vale la seguente relazione:*

$$GH^T = 0 \tag{3.5}$$

Dimostrazione. Dall'equazione (3.4) si ha che

$$c^T = (mG)^T = G^T m^T$$

per ogni messaggio $\mathbf{m}$. Sostituendo questo valore in (3.2), si ha:

$$\begin{aligned} Hc^T &= HG^T m^T = 0 \\ (HG^T)m^T &= 0 \end{aligned} \tag{3.6}$$

dove l'ultimo passaggio segue dalla proprietà associativa del prodotto tra matrici. Siccome (3.6) deve essere valido per ogni $\mathbf{m}$, allora segue che $HG^T = 0$, da cui, calcolando la trasposta, $GH^T = 0$. □

Esponiamo adesso un teorema che permette di calcolare G da H e viceversa.

Teorema 12. *$H = [A|I_{n-k}]$ è una matrice di controllo di parità di un codice lineare $\mathcal{C}[n,k]$ se e solo se $G = [I_k| - A^T]$ è una matrice generatrice dello stesso codice $\mathcal{C}$.*

Dimostrazione. Facciamo vedere che, se $H = [A|I_n - k]$ e $G = [I_k| - A^T]$, allora vale l'equazione (3.5).

$$GH^T = [I_k| - A^T]([A|I_{n-k}])^T = [I_k| - A^T] \left[\frac{A^T}{I_{n-k}}\right] = A^T - A^T = 0.$$

□

La matrice G scritta nella forma $G = [I_k| - A^T]$ è detta *in forma standard*. Osserviamo che se il codice è binario, la matrice G in forma standard diventa $G = [I_k|A^T]$.

Esempio 25. *L'esempio più semplice si ha per codici lineari $\mathcal{C}[n,1]$, definiti su ${\mathbb{F}_2}^n$. Supponiamo che il messaggio $\mathbf{x}$ che si vuole trasmettere sia formato da un unico bit che viene codificato in una parola di codice semplicemente venendo ripetuto per n volte: $\mathbf{x} = 0$ corrisponde a $\mathbf{c}_0 = 0 \dots 0$ e $\mathbf{x} = 1$ a $\mathbf{c}_1 = 1 \dots 1$. In questo caso il decodificatore, se usa il cosiddetto "voto di maggioranza" (ossia decodifica come 0 o come 1 in base a quale bit è più presente) è in grado di correggere fino a $\lfloor (n-1)/2 \rfloor$ errori. Una possibile matrice G per questo codice è:*

$$G = (1|1 \dots 1)$$

Usando il teorema 12 ricaviamo:

$$H = \left(\begin{array}{c|c} 1 & \\ \vdots & I_{n-1} \\ 1 & \end{array}\right)$$

Esempio 26. *È data la seguente matrice di controllo parità H*

$$H = \left(\begin{array}{ccc|ccc} 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \end{array}\right) = (A|I_3)$$

definita per un codice binario, dove $n = 6$ e $k = 3$. Usando di nuovo il teorema 12, ricaviamo la matrice generatrice:

$$G = (I_3|A^T) = \left(\begin{array}{ccc|ccc} 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \end{array}\right)$$

Sia adesso $\mathbf{m} = 011$ il messaggio da codificare. Moltiplicando per la matrice G, otteniamo la codeword $\mathbf{c}$

$$\mathbf{c} = 011 \cdot \left(\begin{array}{ccc|ccc} 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \end{array}\right) = 011011$$

3.4 Codice di Goppa

Il materiale di questa sezione è tratto da Jochemsz [2002].
Il codice di Goppa $\Gamma(L, g(z))$ è un codice lineare definito da:

- un polinomio di grado t in $\mathbb{F}_{p^m}$, per qualche primo p:

$$g(z) = g_0 + g_1 z + \cdots + g_t z^t = \sum_{i=0}^{t} g_i z^i$$

  $g(z)$ è chiamato *polinomio di Goppa*

- un insieme $L \subseteq \mathbb{F}_{p^m}$:

$$L = \{\alpha_1, \ldots \alpha_n\}$$

  che deve essere tale che $g(\alpha_i) \neq 0 \ \forall \alpha_i \in L$.

Dato un vettore $\mathbf{c} = c_1 \ldots c_n$ definito su $\mathbb{F}_p$, gli associamo la funzione

$$R_c(z) = \sum_{i=1}^{n} c_i (z - \alpha_i)^{-1} = \sum_{i=1}^{n} \frac{c_i}{z - \alpha_i} \tag{3.7}$$

dove la scrittura $(z - \alpha_i)^{-1} = \frac{1}{z - \alpha_i}$ indica quel polinomio che è l'inverso di $(z - \alpha_i)$ in modulo $g(z)$.

Definizione 15. *Il codice di Goppa $\Gamma(L, g(z))$ consiste in tutti i vettori $\mathbf{c}$ tali che*

$$R_c(z) = 0 \pmod{g(z)} \tag{3.8}$$

Esempio 27. *Prendiamo il campo finito $\mathbb{F}_{2^4}$, definito tramite il polinomio $f(\alpha) = \alpha^4 + \alpha + 1$. Consideriamo il polinomio $g(z) = z^2 - 1$ e l'insieme $L = \{\alpha^i | 1 \leq i \leq 9\} \subset \mathbb{F}_{2^4}$. Osserviamo subito che $g(\alpha_i) \neq 0 \ \forall \alpha_i \in L$. In questo caso, il codice di Goppa è formato dai vettori $\mathbf{c}$ di lunghezza $n = 9$, per cui si ha:*

$$\sum_{i=1}^{9} c_i (z - \alpha_i)^{-1} = 0 \pmod{z^2 - 1}$$

3.4.1 Parametri del codice di Goppa

In quanto lineare, anche il codice di Goppa è caratterizzato dai tre parametri: n (la lunghezza delle parole), k (la dimensione del codice) e d (la distanza minima). Questi parametri soddisfano le seguenti proprietà:

Proprietà 8. *La dimensione k è tale che $k \geq n - mt$, dove m è l'esponente di $p^m = q$, con q dimensione del campo finito $\mathbb{F}_q$*

Dimostrazione. Abbiamo visto che $(z - \alpha_i)^{-1}$ è un polinomio in modulo $g(z)$:

$$(z - \alpha_i)^{-1} = p_i(z) = p_{i1} + p_{i2} z + \cdots + p_{in} z^{t-1} \pmod{g(z)} \tag{3.9}$$

Allora la formula di $R_c(z)$ si può scrivere come

$$R_c(z) = \sum_{i=1}^{n} c_i (z - \alpha_i)^{-1} = \sum_{i=1}^{n} c_i p_i(z) = \sum_{i=1}^{n} c_i p_{ij}$$

per $1 \leq j \leq t$, dove l'ultima uguaglianza segue se guardiamo gli z^j separatamente. Dall'equazione (3.8) ricaviamo

$$\sum_{i=1}^{n} c_i p_{ij} = 0 \tag{3.10}$$

Abbiamo quindi t equazioni lineari su $\mathbb{F}_q$, che corrispondono ad un numero $\leq mt$ di equazioni su $\mathbb{F}_p$, da cui segue che $k \geq n - mt$. □

Proprietà 9. *La distanza minima d è tale che $d \geq t+1$, con t grado del polinomio $g(z)$*

Dimostrazione. Per dimostrare questa proprietà partiamo dal fatto che per un codice lineare la distanza minima d corrisponde al peso minimo w di una parola di codice diversa da 0. Sia quindi $\mathbf{c} \neq 0$ una parola di codice avente peso w. Sia $c_{\mathrm{i}} \neq 0$ per qualche $i \in \{i_{\mathrm{i}} \dots i_{\mathrm{w}}\}$. Riscriviamo l'equazione (3.8) come frazione:

$$\sum_{i=1}^{n} c_{\mathrm{i}}(z-\alpha_{\mathrm{i}})^{-1} = \sum_{i=1}^{n} \frac{c_{\mathrm{i}}}{(z-\alpha_{\mathrm{i}})} = \frac{\sum_{j=1}^{w} c_{\mathrm{i_j}} \prod_{k=1,k\neq j}^{w}(z-\alpha_{\mathrm{i_k}})}{\prod_{j=1}^{w}(z-\alpha_{\mathrm{i_j}})} = 0 \pmod{g(z)}$$

Sappiamo che $g(z)$ non divide il denominatore, per cui $g(z)$ deve per forza dividere il numeratore. Inoltre, poichè il numeratore ha grado $\leq w-1 = d-1$, allora segue che $d \geq t+1$. □

Parametri in un caso particolare

Un codice con distanza minima d può correggere fino a r errori se $2r+1 \leq d$. Vediamo adesso un caso in cui il limite inferiore di r si può aumentare.

Teorema 13. *Sia $g(z)$ un polinomio separabile, cioè senza radici di molteplicità più grande di uno, definito su $GF(2^m)$. Allora il codice di Goppa $\Gamma(L, g(z))$ ha distanza minima $d \geq 2t+1$, dove t è il grado di $g(z)$.*

Dimostrazione. Sia $\mathbf{c}$ una parola di codice avente peso minimo w. Sappiamo dalla dimostrazione precedente che $\sum_{i=1}^{n} \frac{c_{\mathrm{i}}}{z-\alpha_{\mathrm{i}}} = 0 \pmod{g(z)}$ se e solo se $g(z)$ divide $f(z)$, con

$$f(z) = \sum_{j=1}^{w} c_{\mathrm{i_j}} \prod_{\substack{k=1 \\ k\neq j}}^{w}(z-\alpha_{\mathrm{i_k}}) = \sum_{j=1}^{w} \prod_{\substack{k=1 \\ k\neq j}}^{w}(z-\alpha_{\mathrm{i_k}})$$

perchè il codice è binario. Notiamo inoltre che l'ultima espressione è la derivata di

$$\prod_{j=1}^{w}(z-\alpha_{\mathrm{i_j}})$$

ed una derivata binaria può avere solo esponenti pari: $f(z) = f_0 + f_2 z^2 + \cdots + f_{2u} z^{2u}$, con $2u \leq w-1$, che in $GF(2^m)$ è equivalente a

$$f(z) = (k_0 + k_2 z + \cdots + k_2 u z^u)^2 = (k(z))^2$$

con $k_i^2 = f_i$ e $2u \leq w-1$. Da qui segue che $g(z)$ divide $(k(z))^2$, con $k(z)$ polinomio di grado u e $2u \leq w-1$. Siccome g non ha radici doppie, deve anche dividere $k(z)$, da cui ricaviamo che $t \leq u$ e che $w-1 \geq 2u \geq 2t$, da cui $d \geq 2t+1$. □

Lemma 1. *Sia $\Gamma(L, g(z))$ un codice binario di Goppa con $g(z)$ separabile. Allora $\Gamma(L, g(z)) = \Gamma(L, g^2(z))$*

Dimostrazione. Nell'ultima dimostrazione abbiamo fatto vedere che $g(z)$ divide $k(z)$, il che implica che $g^2(z)$ divide $k^2(z) = f(z)$. Da questo segue che: $\mathbf{c}$ è una parola di codice in $\Gamma(L, g^2(z))$ $\Leftrightarrow$ $g^2(z)$ divide $f(z)$ $\Leftrightarrow$ $g(z)$ divide $f(z)$ $\Leftrightarrow$ $\mathbf{c}$ è una parola di codice di $\Gamma(L, g(z))$.

□

Useremo il teorema 13 e il lemma 1 nella sezione 3.4.4, quando parleremo dell'algoritmo di correzione del codice di Goppa nel caso particolare.

3.4.2 Matrice di controllo parità e matrice generatrice

Sappiamo che vale l'equazione (3.10)

$$\sum_{i=1}^{n} c_{\mathrm{i}} p_{\mathrm{ij}} = 0$$

dove i c_i sono le componenti della parola ci codice $\mathbf{c}$ e i p_{ij} sono i coefficienti del polinomio $(z-\alpha_i)^{-1}$ definiti nella formula 3.9. Ma allora la matrice di controllo parità H sarà data da:

$$H = \begin{pmatrix} p_{1,1} & \cdots & p_{n,1} \\ p_{1,2} & \cdots & p_{n,2} \\ \vdots & \ddots & \vdots \\ p_{1,n} & \cdots & p_{n,n} \end{pmatrix}$$

Per calcolare i fattori p_{ij} riscriviamo il polinomio p_{i} come:

$$p_{\mathrm{i}}(z) = (z-\alpha_{\mathrm{i}})^{-1} = -\frac{g(z)-g(\alpha_{\mathrm{i}})}{z-\alpha_{\mathrm{i}}} \cdot g(\alpha_{\mathrm{i}})^{-1}$$

Questo si può fare perché:

$$-(z-\alpha_{\mathrm{i}}) \cdot \frac{g(z)-g(\alpha_{\mathrm{i}})}{z-\alpha_{\mathrm{i}}} \cdot g(\alpha_{\mathrm{i}})^{-1} = -g(z)g(\alpha_{\mathrm{i}}) + 1 = 1 \pmod{g(z)}$$

e $p_{\mathrm{i}}(z)$ è l'unico polinomio per cui $p_{\mathrm{i}}(z) \cdot (z-\alpha_{\mathrm{i}}) = 1 \pmod{g(z)}$. Se sostituiamo $g(z) = g_0 + g_1 z + \cdots + g_{\mathrm{t}} z^t$ e mettiamo $h_{\mathrm{i}} = g(\alpha_{\mathrm{i}})^{-1}$, otteniamo:

$$\begin{aligned} p_{\mathrm{i}}(z) &= -\frac{g_0 - g_0 + g_1 z - g_1\alpha_{\mathrm{i}} + \cdots + g_{\mathrm{t}} z^t - g_{\mathrm{t}}\alpha_{\mathrm{i}}^t}{z-\alpha_{\mathrm{i}}} \cdot h_{\mathrm{i}} = \\ &= -\frac{g_{\mathrm{t}}(z^t - \alpha_{\mathrm{i}}^t) + \cdots + g_1(z-\alpha_{\mathrm{i}})}{z-\alpha_{\mathrm{i}}} \cdot h_{\mathrm{i}} = \\ &= -[g_{\mathrm{t}}(z^{t-1} + z^{t-2}\alpha_{\mathrm{i}} + \cdots + \alpha_{\mathrm{i}}^{t-1}) + g_{\mathrm{t}-1}(z^{t-2} + z^{t-3}\alpha_{\mathrm{i}} + \cdots + \alpha_{\mathrm{i}}^{t-2}) + \cdots + g_2(z+\alpha_{\mathrm{i}}) + g_1]\dot{h}_{\mathrm{i}} \end{aligned} \tag{3.11}$$

D'altra parte, $p_{\mathrm{i}}(z)$ può essere visto come

$$p_{\mathrm{i}}(z) = p_{\mathrm{i}1} + p_{\mathrm{i}2} z + \cdots + p_{\mathrm{it}} z^{t-1} \tag{3.12}$$

Mettendo insieme le equazioni (3.11) e (3.12) otteniamo il sistema

$$\begin{cases} p_{\mathrm{i}1} = -[g_{\mathrm{t}}\alpha_{\mathrm{i}}^{t-1} + g_{\mathrm{t}-1}\alpha_{\mathrm{i}}^{t-2} + \cdots + g_2\alpha_{\mathrm{i}} + g_1] \cdot h_{\mathrm{i}} \\ p_{\mathrm{i}2} = -[g_{\mathrm{t}}\alpha_{\mathrm{i}}^{t-2} + g_{\mathrm{t}-1}\alpha_{\mathrm{i}}^{t-3} + \cdots + g_2] \cdot h_{\mathrm{i}} \\ \vdots \\ p_{\mathrm{i},\mathrm{t}-1} = -[g_{\mathrm{t}}\alpha_{\mathrm{i}} + g_{\mathrm{t}-1}] \cdot h_{\mathrm{i}} \\ p_{\mathrm{it}} = -g_{\mathrm{t}} h_{\mathrm{i}} \end{cases}$$

la cui risoluzione porta al calcolo dei coefficienti p_{ij}. In forma matriciale, possiamo scrivere $H = CXY$, con

$$C = \begin{pmatrix} -g_{\mathrm{t}} & -g_{\mathrm{t}-1} & g_{\mathrm{t}-2} & \cdots & -g_1 \\ 0 & -g_{\mathrm{t}} & -g_{\mathrm{t}-1} & \cdots & -g_2 \\ 0 & 0 & -g_{\mathrm{t}} & \cdots & -g_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & -g_{\mathrm{t}} \end{pmatrix} \tag{3.13}$$

$$X = \begin{pmatrix} \alpha_1^{t-1} & \alpha_2^{t-1} & \cdots & \alpha_n^{t-1} \\ \alpha_1^{t-2} & \alpha_2^{t-2} & \cdots & \alpha_n^{t-2} \\ \vdots & \vdots & \vdots & \vdots \\ \alpha_1 & \alpha_2 & \cdots & \alpha_n \\ 1 & 1 & \cdots & 1 \end{pmatrix} \tag{3.14}$$

$$Y = \begin{pmatrix} h_1 & 0 & \cdots & 0 \\ 0 & h_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & h_n \end{pmatrix} \tag{3.15}$$

Per calcolare la matrice G basta invece considerare che $GH^T = 0$.

Esempio 28. *Riprendiamo il campo $\mathbb{F}_{16}$ già visto nell'esempio 17, considerando il polinomio irriducibile $f(x) = x^4 + x + 1$. Abbiamo già fatto vedere che x è un generatore del campo. Per questo esempio consideriamo come generatore α, per cui vale una tabella analoga a quella dell'esempio 3.1. Prendiamo il codice di Goppa $\Gamma(g(z), L)$, con*

$$g(z) = (z+\alpha)(z+\alpha^{14}) = z^2+\alpha^{14}z+\alpha z+\alpha^{15} = z^2+(\alpha^{14}+\alpha)z+1 = z^2+(1+\alpha+\alpha^3)z+1 = z^2+\alpha^7 z+1$$

$$L = \{\alpha^i | i = 2,3,\cdots,13\}$$

Per calcolare H, sfruttiamo le matrici (3.13), (3.14) e (3.15), con $g_1 = \alpha^7$, $g_2 = 1$ e con $\alpha_1 = \alpha^2, \alpha_2 = \alpha^3, \cdots, \alpha_{12} = \alpha^{13}$. I fattori h_i vengono invece così calcolati:

$$h_1 = g(\alpha_1)^{-1} = g(\alpha^2)^{-1} = (\alpha^4 + \alpha^9 + 1)^{-1} = (1 + \alpha + \alpha + \alpha^3 + 1)^{-1} = \alpha^{-3} = \alpha^{12}$$

$$h_2 = g(\alpha_2)^{-1} = g(\alpha^3)^{-1} = (\alpha^6+\alpha^{10}+1)^{-1} = (\alpha^2+\alpha^3+1+\alpha+\alpha^2+1)^{-1} = (\alpha+\alpha^3)^{-1} = \alpha^{-9} = \alpha^6$$

e così via. La matrice H risulta quindi essere così formata:

$$H = \begin{pmatrix} \alpha^9 & \alpha^{10} & \alpha^9 & \alpha^{14} & \alpha^6 & 0 & \alpha^{10} & \alpha^8 & \alpha^2 & \alpha^7 & \alpha^{14} & \alpha^6 \\ \alpha^{12} & \alpha^6 & \alpha^6 & \alpha & \alpha^{11} & 1 & \alpha^{14} & \alpha^8 & \alpha^{11} & \alpha^{14} & \alpha^{12} & \alpha \end{pmatrix}$$

Imponendo poi $GH^T = 0$, si ricava:

$$G = \begin{pmatrix} 1 & \alpha^{13} & \alpha^9 & \alpha^{14} & \alpha^5 & \alpha^6 & \alpha^{10} & \alpha^{12} & \alpha & \alpha^3 & \alpha^2 & 1 \end{pmatrix}$$

3.4.3 Codifica e decodifica di un messaggio

La codifica di un messaggio **m** consiste semplicemente nel dividerlo in blocchi di lunghezza k ed eseguire poi la motiplicazione con la matrice G.

$$m_1 \cdots m_k \cdot G = c_1 \cdots c_n.$$

La decodifica segue lo stesso principio, scrivendo però l'equazione come:

$$G^T \cdot \begin{pmatrix} m_1 \\ \vdots \\ m_k \end{pmatrix} = \begin{pmatrix} c_1 \\ \vdots \\ c_n \end{pmatrix}$$

Per trovare il vettore dei messaggi, si parte dalla matrice

$$\left(\begin{array}{c|c} & c_1 \\ G^T & \vdots \\ & c_n \end{array}\right)$$

e si opera una serie di passaggi elementari per ottenere la matrice equivalente

$$\left(\begin{array}{c|c} & m_1 \\ I_k & \vdots \\ & m_k. \\ \hline \multicolumn{2}{c}{\mathrm{P}} \end{array}\right) \tag{3.16}$$

3.4.4 Correzione di errori

Sia $\mathbf{y} = y_1 \cdots y_\mathrm{n} = c_1 \cdots c_\mathrm{n} + e_1 \cdots e_\mathrm{n}$ il messaggio ricevuto, dove il vettore $\mathbf{e} = e_1 \cdots e_\mathrm{n}$ rappresenta il termine di errore. Supponiamo che $\mathbf{y}$ contega al più r errori, con $2r + 1 \leq d$ o $r \leq \lfloor t/2 \rfloor$ nel caso in cui $d = t + 1$. In altre parole, $e_\mathrm{i} \neq 0$ in esattamente r posizioni.
Per correggere la parola ricevuta dobbiamo trovare $\mathbf{e}$; per trovare $\mathbf{e}$ abbiamo bisogno di:

- $B = \{i | e_\mathrm{i} \neq 0\} = \{i_1, \cdots, i_\mathrm{r}\}$, insieme delle posizioni degli errori
- i corrispondenti valori e_i, con $i \in B$

A questo scopo, definiamo i seguenti polinomi: *error locator* $\sigma(z)$ e *error evaluator* $\omega(z)$

$$\sigma(z) \triangleq \prod_{i \in B} (z - \alpha_\mathrm{i})$$

$$\omega(z) \triangleq \sum_{i \in B} \prod_{\substack{j \in B \\ j \neq i}} (z - \alpha_\mathrm{j})$$

Osserviamo inoltre che l'insieme B può essere ridefinito come $B = \{i | \alpha_\mathrm{i} \text{ è una radice di } \sigma(z)\}$.
Definiamo anche la *sindrome* $s(z)$ della parola ricevuta come:

$$s(z) \triangleq \sum_{i=1}^{n} \frac{y_\mathrm{i}}{z - \alpha_\mathrm{i}} = \sum_{i=1}^{n} \frac{c_\mathrm{i} + e_\mathrm{i}}{z - \alpha_\mathrm{i}} = \sum_{i=1}^{n} \frac{c_\mathrm{i}}{z - \alpha_\mathrm{i}} + \sum_{i \in B} \frac{e_\mathrm{i}}{z - \alpha_\mathrm{i}} = \sum_{i \in B} \frac{e_\mathrm{i}}{z - \alpha_\mathrm{i}} \pmod{g(z)}$$

Queste funzioni soddisfano le seguenti proprietà:

1. $\sigma(z)$ ha grado r
2. $\omega(z)$ ha grado al più $r - 1$
3. $mcd(\sigma(z), \omega(z)) = 1$
4. $e_k = \frac{\omega(\alpha_k)}{\sigma'(\alpha_k)}$, con $k \in B$
5. $\sigma(z)s(z) = \omega(z) \pmod{g(z)}$

Dimostrazione. Le prime tre proprietà seguono direttamente dalla definizione. Per quanto riguarda la proprietà 4, abbiamo che

$$\sigma'(z) = \sum_{i \in B} \prod_{\substack{j \in B \\ j \neq i}} (z - \alpha_{\mathrm{j}})$$

Calcoliamo il rapporto $\frac{\omega(\alpha_{\mathrm{k}})}{\sigma'(z)}$, con $k \in B$:

$$\frac{\omega(\alpha_{\mathrm{k}})}{\sigma'(z)} = \frac{\sum_{i \in B} e_{\mathrm{i}} \prod_{j \in B, j \neq i} (\alpha_k - \alpha_{\mathrm{j}})}{\sum_{i \in B} \prod_{j \in B, j \neq i} (z - \alpha_{\mathrm{j}})} = e_{\mathrm{k}}$$

in quanto in ogni termine della produttoria al numeratore è presente $(\alpha_k - \alpha_k)$ a parte in quello con $i = k$.
Per quanto riguarda la proprietà 5, abbiamo:

$$\sigma(z)s(z) = \prod_{i \in B} (z - \alpha_i) \sum_{i \in B} \frac{e_{\mathrm{i}}}{(z - \alpha_{\mathrm{i}})} \pmod{g(z)}$$

Mettiamo $B = \{1,2,\cdots,r\}$ e troviamo:

$$\begin{aligned}
&[(z-\alpha_1)(z-\alpha_2)\cdots(z-\alpha_r)]\left(\frac{e_1}{z-\alpha_1} + \frac{e_2}{z-\alpha_2} + \cdots + \frac{e_r}{z-\alpha_r}\right) = \\
&= e_1(z-\alpha_2)(z-\alpha_3)\cdots(z-\alpha_r) + e_2(z-\alpha_1)(z-\alpha_3)\cdots(z-\alpha_r) + \cdots + e_r(z-\alpha_1)(z-\alpha_2)\cdots(z-\alpha_{r-1}) = \\
&= \sum_{i \in B} e_i \prod_{\substack{j \in B \\ j \neq i}} (z - \alpha_j) = \omega(z)
\end{aligned}$$

□

Per correggere una parola di codice, bisogna risolvere l'equazione

$$\sigma(z)s(z) = \omega(z) \pmod{g(z)}$$

dove $g(z)$ è noto e $s(z)$ si può calcolare:

$$\begin{aligned}
\sigma(z) &= \sigma_0 + \sigma_1 z + \cdots + \sigma_{r-1} z^{r-1} + z^r \\
\omega(z) &= \omega_0 + \omega_1 z + \cdots + \omega_{r-1} z^{r-1}
\end{aligned}$$

Otteniamo quindi un sistema con t equazioni e $2r$ incognite. Poichè $2t \leq r$, il sistema ha un unica soluzione.

Algoritmo per la correzione di errori

Presentiamo adesso un algoritmo in grado di correggere fino a $\lfloor \frac{t}{2} \rfloor$ errori in un codice di Goppa. Sia $y_1 \cdots y_n$ una parola di codice ricevuta contenente r errori, con $2r \leq t$. L'algoritmo procede in questo modo:

1. Calcola la sindrome:

$$s(z) = \sum_{i=1}^{n} \frac{y_{\mathrm{i}}}{z - \alpha_i}$$

2. Risolve l'equazione $\sigma(z)s(z) = \omega(z) \pmod{g(z)}$, scrivendo:
$$\sigma(z) = \sigma_0 + \sigma_1 z + \cdots + \sigma_{r-1} z^{r-1} + z^r$$
$$\omega(z) = \omega_0 + \omega_1 z + \cdots + \omega_{r-1} z^{r-1}$$
Inoltre, se il codice è binario $\omega(z) = \sum_{i \in B} 1 \cdot \prod_{\substack{j \in B \\ j \neq i}} = \sigma'(z)$
3. Determina l'insieme di error locations $B = \{i | \sigma(i) = 0\}$
4. Calcola i valori degli errori $e_{\mathrm{i}} = \frac{\omega(\alpha_i)}{\sigma'(\alpha_i)}$, con $i \in B$
5. Assembla il vettore degli errori $e_1, \cdots e_n$, che è dato da e_{i} $\forall i \in B$ e 0 altrimenti
6. Ricostruisce la parola originale $\mathbf{c} = \mathbf{y} - \mathbf{e}$

Algoritmo di correzione nel caso particolare

Prendiamo il codice Goppa $\Gamma(L, g(z))$ definito su $GF(2^m)$ e tale che $g(z)$ è un polinomio separabile di grado t. Abbiamo fatto vedere nel teorema 13 che la distanza minima $d \geq 2t+1$; dobbiamo quindi essere in grado di correggere fino a t errori. Se però consideriamo che $\Gamma(L, g(z)) = \Gamma(L, g^2(z))$ e che $g^2(z)$ ha grado $2t$, allora siamo in grado di correggere fino a $\lfloor \frac{2t}{2} \rfloor = t$ errori. Per fare questo occorre però come prima cosa calcolare la matrice di parità $\hat{H}$ di $\Gamma(L, g^2(z))$, che useremo per ricavare la sindrome.

Esempio 29. *Riprendiamo lo stesso codice di Goppa già visto negli esempi 28 e 17, cioè quello con:*
$$g(z) = (z + \alpha)(z + \alpha^{14}) = z^2 + \alpha^7 z + 1$$
$$L = \{\alpha^i | 2 \leq i \leq 13\}$$
defintio su $GF(2^4)$. I parmetri di questo codice sono $n = 12, d \geq 5, k \geq 4$. Supponiamo quindi che tale codice venga usato per codificare il vettore $\mathbf{x} = (1,1,1,1)$
$$\mathbf{c} = (1,1,1,1) \cdot \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix} = (1,1,0,0,0,0,1,0,1,1,1,1)$$
Durante la trasmissione, vengono commessi due errori negli ultimi due bit, ottenendo quindi il vettore
$$\mathbf{y} = (1,1,0,0,0,0,1,0,1,1,0,0)$$
Il destinatario, sfruttando il fatto che $g(z)$ è separabile ed è definito su un campo binario, può usare l'algoritmo di correzione nel caso particolare. Per prima cosa, quindi, calcola
$$\hat{g}(z) = g^2(z) = (z^2 + \alpha^7 z + 1)^2 = z^4 + \alpha^{14} z^2 + 1$$
I fattori $\hat{h}_i$ sono dati da: $\hat{h}_i = (g^2(\alpha_i))^{-1} = (g(\alpha_i))^{-1^2} = h_{\mathrm{i}}^2$ e la matrice $\hat{H}$ sarà quindi:
$$\hat{H} =, \begin{pmatrix} \alpha^5 & \alpha^8 & \alpha^7 & \alpha^3 & \alpha^3 & 0 & \alpha^{13} & \alpha^{10} & \alpha^{14} & \alpha^{10} & \alpha^{10} & \alpha^{10} \\ \alpha^3 & \alpha^5 & \alpha^3 & \alpha^{13} & \alpha^{12} & 0 & \alpha^5 & 1 & \alpha^4 & \alpha^{14} & \alpha^{13} & \alpha^{12} \\ \alpha^{11} & 1 & \alpha & \alpha^7 & \alpha^{13} & \alpha^7 & \alpha^6 & \alpha^{10} & \alpha^2 & \alpha^9 & \alpha^6 & 1 \\ \alpha^9 & \alpha^{12} & \alpha^{12} & \alpha^2 & \alpha^7 & 1 & \alpha^{13} & \alpha & \alpha^7 & \alpha^{13} & \alpha^9 & \alpha^2 \end{pmatrix}$$
Usiamo l'algortimo visto nella sezione 3.4.4 per correggere $\mathbf{y}$.

1. *Calcoliamo la sindrome*

$$s(z) = \sum_{i=1}^{9} \frac{y_i}{z - \alpha_i} =$$
$$= \frac{1}{z-\alpha^2} + \frac{1}{z-\alpha^3} + \frac{1}{z-\alpha^8} + \frac{1}{z-\alpha^{10}} + \frac{1}{z-\alpha^{11}} =$$
$$= (\alpha^5 + \alpha^8 + \alpha^{13} + \alpha^{14} + \alpha^{10}) + (\alpha^3 + \alpha^4 + \alpha^{14})z + (\alpha^{11} + 1 + \alpha^6 + \alpha^2 + \alpha^9)z^2 +$$
$$+ (\alpha^9 + \alpha^{12} + \alpha^7 +)z^3 = \alpha z + \alpha^{13} z^2 + \alpha^{11} z^3$$

2. *Calcoliamo* $\sigma(z)s(z) \pmod{z^4 + \alpha^{14} z^2 + 1}$

$$\sigma(z)s(z) = (z^2 + \sigma_1 z + \sigma_0)(\alpha^{11} z^3 + \alpha^{13} z^2 + \alpha z) =$$
$$= \alpha^{11} z^5 + (\alpha^{13} + \alpha^{11}\sigma_1) z^4 + (\alpha + \alpha^{13}\sigma_1 + \alpha^{11}\sigma_0) z^3 + (\alpha\sigma_1 + \alpha^{13}\sigma_0) z^2 + \alpha\sigma_0 z =$$
$$= (\alpha + \alpha^{13}\sigma_1 + \alpha^{11}\sigma_0 + \alpha^{10}) z^3 + (\alpha\sigma_1 + \alpha^{13}\sigma_0 + \alpha^{12} + \alpha^{10}\sigma_1) z^2 + (\alpha\sigma_0 + \alpha^{11}) z + (\alpha^{13} + \alpha^{11}\sigma_1)$$
$$\pmod{z^4 + \alpha^{14} z + 1}$$

Il sistema da risolvere sarà quindi

$$\begin{cases} \sigma_1 = \alpha^{13} + \alpha^{11}\sigma_1 \\ 0 = \alpha\sigma_0 + \alpha^{11} \\ 0 = \alpha\sigma_1 + \alpha^{13}\sigma_0 + \alpha^{12} + \alpha^{10}\sigma_1 \\ 0 = \alpha + \alpha^{13}\sigma_1 + \alpha^{11}\sigma_0 + \alpha^{10} \end{cases}$$

I risultati sono $\sigma_0 = \alpha^{10}, \sigma_1 = \alpha$*; perciò:*

$$\sigma(z) = z^2 + \alpha z + \alpha^{10} = (z + \alpha^{12})(z + \alpha^{13})$$

3. *Poichè* $\alpha_{11} = \alpha^{12}$ *e* $\alpha_{12} = \alpha^{13}$*,* $B = \{i | \sigma(\alpha_i) = 0\} = \{11{,}12\}$

4. $\mathbf{e} = (0{,}0{,}0{,}0{,}0{,}0{,}0{,}0{,}0{,}0{,}1{,}1)$

5. $\mathbf{c} = \mathbf{y} - \mathbf{e} = (1{,}1{,}0{,}0{,}0{,}0{,}1{,}0{,}1{,}1{,}1{,}1)$

Usando le matrici, possiamo risalire al messaggio originale:

$$[G^T | c^T] = \left(\begin{array}{cccc|c} 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{array}\right) \sim \left(\begin{array}{cccc|c} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ \hline 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{array}\right)$$

Il messaggio originale è quindi (1,1,1,1)*.*

3.5 Sistema crittografico di McElice

Questa sezione è tratta da Jochemsz [2002].
Per costruire il sistema crittografico di McEliece bisogna scegiere:

- un polinomio $g(z)$ separabile su $GF(2^m)$ di grado t; il codice di Goppa $\Gamma(L, g(z))$ avrà dimensione $k \geq n - mt$ e distanza minima $d \geq 2t + 1$
- una matrice casuale S di dimensione $k \times k$ densa (cioè che abbia la maggior parte degli elementi non nulli) e non singolare (cioè con determinante diverso da 0)
- una matrice casuale di permutazione P di dimensione $n \times n$

Dopo aver calcolato la matrice generatrice G del codice di Goppa, si calcola $G*$:

$$G* = SGP$$

Quello di McEliece è un sistema crittografico a chiave pubblica, in cui la coppia $G*, t$ è nota, mentre rimangono privati $g(z), G, S, P$. Il messaggio è diviso in m_k stringhe binarie di lunghezza k; per ogni stringa, prendiamo un errore random di lunghezza n con al più t coordinate uguali a 1, in modo che $w(e) \leq t$. Per cifrare si calcola $\mathbf{y} = \mathbf{m}G^* + \mathbf{e}$ e lo si spedisce. Il destinatario usa la sua matrice segreta P per calcolare $\mathbf{y}'$:

$$\begin{aligned}
&\mathbf{y}' = \mathbf{y}P^{-1} = \\
&= (\mathbf{m}G* + \mathbf{e})P^{-1} = \\
&= \mathbf{m}G* P^{-1} + \mathbf{e}P^{-1} = \\
&= \mathbf{m}SGPP^{-1} + \mathbf{e}' = \\
&= (\mathbf{m}S)G + \mathbf{e}' = \\
&= \mathbf{m}'G + \mathbf{e}'
\end{aligned}$$

Il destinatario usa l'algoritmo di correzione del codice di Goppa per eliminare $\mathbf{e}'$. Esegue poi la decodifica per trovare $\mathbf{m}'$, da cui calcola $\mathbf{m}$ moltiplicando per S^{-1}.
McEliece propone di usare questi valori: $m = 10$, $t = 50$, $n = 2^{10} = 1024$, da cui ricaviamo $k \geq 1024 - 10 \cdot 50 = 525$.

Esempio 30. *Usiamo lo stesso codice di Goppa degli esempi precedenti. Ricordiamo che questo codice può correggere fino a due errori. Per il sistema di McEliece dobbiamo scegliere le matrici S e P che modificano la matrice G*

$$G = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Scegliamo S e P come:

$$S = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

$$P = \begin{pmatrix} 0&0&0&1&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&1&0 \\ 0&0&0&0&1&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0&0&0&0&0 \\ 1&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&1&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&1&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&1&0&0 \\ 0&0&1&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&0&0&1&0&0&0&0 \end{pmatrix}$$

In questo caso, la matrice $G*$ diventa

$$G* = SGP = \begin{pmatrix} 1&0&1&1&0&1&1&1&0&1&0&0 \\ 1&1&0&0&0&1&0&1&1&0&0&0 \\ 1&0&1&1&1&0&1&1&1&0&1&0 \\ 0&0&1&1&0&0&0&1&1&1&1&1 \end{pmatrix}$$

Conoscendo questa matrice, ogni utente può cifrare delle informazioni. Supponiamo ad esempio di voler cifrare il messaggio $\mathbf{m} = (0,1,0,1)$. *Per prima cosa calcoliamo*

$$\mathbf{m}G* = (1,1,1,1,0,1,0,0,0,1,1,1)$$

e aggiungiamo un errore random $\mathbf{e} = (0,1,1,0,0,0,0,0,0,0,0,0)$. *Calcoliamo poi* $\mathbf{y}$ *e lo trasmettiamo:*

$$\mathbf{y} = \mathbf{m}G* + \mathbf{e} = (1,0,0,1,0,1,0,0,0,1,1,1)$$

Il destinatario calcola $\mathbf{y}' = yP^{-1}$ *usando la sua matrice segreta* P

$$\mathbf{y}' = \mathbf{m}SG + \mathbf{e}' = (1,1,0,0,1,1,1,0,1,0,0,0)$$

Usando l'algoritmo di correzione degli errori, si trova

$$\mathbf{m}SG = (1,1,0,1,1,1,1,0,1,1,0,0)$$

da cui si ricava $\mathbf{m}S$. *Facendo delle operazioni sulla matrice* $[G^T|(mS)^T]$ *che abbiamo visto nell'equazione* (3.16) *si trova:*

$$[G^T|(mS)^T] = \left(\begin{array}{cccc|c} 1&0&0&0&1 \\ 1&1&0&1&1 \\ 1&0&1&0&0 \\ 1&0&0&1&1 \\ 0&1&1&0&1 \\ 1&1&0&0&1 \\ 0&1&1&1&1 \\ 1&1&1&1&0 \\ 0&0&1&0&1 \\ 1&0&0&0&1 \\ 0&1&1&0&0 \\ 0&0&0&1&0 \end{array}\right) \sim \left(\begin{array}{cccc|c} 1&0&0&0&1 \\ 0&1&0&0&0 \\ 0&0&1&0&1 \\ 0&0&0&1&0 \\ \hline 1&1&0&1&1 \\ 1&1&0&1&1 \\ 1&0&1&0&0 \\ 1&0&0&1&1 \\ 0&0&1&0&1 \\ 0&1&1&1&1 \\ 1&1&1&1&0 \\ 1&0&0&0&1 \end{array}\right)$$

da cui deriva $mS = (1,0,1,0)$. *Moltiplichiamo infine per* S^{-1} *e troviamo:*

$$\mathbf{m} = (1,0,1,0) \cdot \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{pmatrix} = (0,1,0,1)$$

che corrisponde al messaggio che era stato inviato.

3.5.1 Attacchi al sistema di McEliece

Indovinare le matrici S e P

Se un crittanalista indovina S e P, è in grado di ricavare G, da cui a sua volta può risalire a $g(z)$, e rompere il codice. Vediamo qual è in concreto la possibilità di indovinare S e P.
Iniziamo con S, che sappiamo essere una matrice $k \times k$ invertibile; il che significa che le sue righe sono linearmente indipendenti. Dati questi vincoli, quante matrici S si possono fare?

- la prima riga r_1 di S può essere formata in $2^k - 1$ modi diversi (l'unico caso non ammesso è quello con tutti zeri)
- la seconda riga r_2, può essere costruita in $2^k - 2$ modi (viene esclusa anche la riga $1 \cdot r_1$)
- per la terza riga r_3 in $2^k - 4$ modi (non sono valide: la riga con tutti zeri, r_1, r_2, $r_1 + r_2$)

e così via. Il numero totale di matrici binarie $k \times k$ sarà quindi

$$\prod_{i=0}^{k-1} (2^k - 2^i)$$

La matrice P è invece una matrice di permutazione di dimensione $n \times n$, e sappiamo che il totale di tali matrici è $n!$.
Nel caso dei parametri proposti da McEliece, le possibilità di trovare la S con un tentativo casuale sono quindi

$$\frac{1}{\prod_{i=0}^{523}(2^{254} - 2^i)} = 0{,}8459238718 \cdot 10^{-82655}$$

mentre quella di trovare P è

$$\frac{1}{1024!} = 0{,}1845519398 \times 10^{-2639}$$

Confronto tra parole di codice

Un altro possibile approccio di tipo "forza bruta" consiste nel generare tutte le possibili 2^k parole di codice per il codice definito da $G*$ con lo scopo di trovare quell'unica parola del codice $\mathbf{c}$ che è più vicina ad $\mathbf{y}$, in quanto $\mathbf{c} = \mathbf{m}G*$ e $d(\mathbf{c}, \mathbf{y}) \leq t$. Usando i parametri proposti da McEliece si ha che occorre fare $2^{254} = 0{,}5491838128 \cdot 10^{158}$ tentativi per trovare tale parola del codice.

Decodifica della sindrome

Il crittoanalista calcola $H*$, la matrice di controllo parità associata a $G*$ grazie alla formula $(G*) \cdot (H*)^T = 0$. Questa matrice ha rango $n - k$. Si può quindi calcolare la sindrome della parola trasmessa $\mathbf{y}$:

$$\mathbf{y}(H*)^T = (\mathbf{m}G* + \mathbf{e})(H*)^T = \mathbf{m}G*(H*)^T + \mathbf{e}(H*)^T = \mathbf{e}(H*)^T$$

Trovando il vettore $\mathbf{e}$ corretto, il crittanalista scopre $\mathbf{m}G*$ e può facilmente calcolare m applicando l'eliminazione gaussiana. Inoltre, può generare tutti i possibili vettori di errore di lunghezza n e peso $\leq t$, calcolare $e(H*)^T$ e confrontare questi con la sindrome. Il numero di vettori da testare è

$$\sum_{i=0}^{50} \binom{1024}{i} = 0{,}3362308934 \cdot 10^{86}$$

Cifratura multipla dello stesso messaggio

Uno degli svantaggi del criptosistema di McEliece consiste nel fatto che non è un sistema sicuro per mandare più volte lo stesso messaggio con la stessa matrice di cifratura $G*$. Infatti, consideriamo due diverse cifrature del messaggio $\mathbf{m}$:

$$\mathbf{y} = \mathbf{m}G* + e$$

$$\mathbf{y}' = \mathbf{m}G* + e'$$

Se i due errori sono scelti casualmente, ci si aspetta che, in media:

- $e_i = 0$ e $e'_i = 0$ in $\frac{(n-t)^2}{n}$ posizioni;
- $e_i = 0$ e $e'_i = 1$ (o viceversa) in $\frac{2t(n-t)}{n}$ posizioni
- $e'_i = 1$ e $e'_i = 1$ in $\frac{t^2}{n}$ posizioni

Con i parametri standard di McEliece, ci si aspetta che sia e_i sia e'_i siano uguali a uno in

$$\frac{50^2}{1024} \approx 2{,}441406250$$

coordinate, che corrispondono a circa tre coordinate sul numero totale di coordinate in cui i due errori coincidono, che è dato da:

$$\frac{(1024-50)^2}{1024} + \frac{50^2}{1024} \approx 929$$

Per trovare il messaggio m, occorre un insieme con $k = 524$ coordinate dove $\mathbf{y}$ e $\mathbf{y}'$ sono senza errori e su cui $G*$ abbia un rank completo. In questo caso, la possibilità che una scelta casuale di 524 coordinate sia senza errori è pari a:

$$\frac{\binom{926}{3}\binom{3}{0}}{\binom{929}{3}} \approx 0{,}08250837050$$

e perciò ci aspettiamo che, seguendo questo approccio, un attacco abbia sucesso dopo appena

$$\frac{1}{0{,}08250837050} \approx 12{,}11998242$$

tetativi. Per questi motivi diciamo che il McEliece non è sicuro per criptare più volte lo stesso messaggio.

3.5.2 Vantaggi e svantaggi

Concludiamo il capitolo con una breve panoramica dei vantaggi e degli svantaggi del sistema di McEliece. I principali vantaggi sono:

- è un sistema facile da capire e da studiare, la cui sicurezza è stata studiata a lungo fin dalla sua introduzione nel 1978;
- preenza di un fattore di casualità, diversamente da quanto accade in alcuni sistemi classici come RSA.
- è ritenuto uno dei sistemi sicuri anche con l'avvento del computer quantistico, in quanto non basato sulla fattorizzazione Chen et al. [2016] e per il già citato fattore di casualità introdotto Roering [2013].

Gli svantaggi invece sono:

- è più facile da rompere quando si cifra più di una volta lo stesso messaggio,
- i messaggi cifrati sono n/k volte più lunghi di quelli in chiaro
- le chiavi hanno dimensione più grande di quelle usate in altri crittosistemi, come ad esempio in RSA
- allo stato attuale, non può essere usato per gli schemi di firma (come invece accade per RSA e per altri sistemi classici)

Con l'aumento dello spazio di memoria disponibile ad un prezzo contenuto, quello della dimensione della chiave non sembra un grosso problema. L'unico vero ostacolo rimane quindi il fatto di non poter essere usato per gli schemi di firma, di cui diamo una breve spiegazione, tratta da Courtois et al. [2001].
Negli schemi di firma digitale, il mittente calcola un hash del documento e ottiene un valore y. Usando la sua chiave privata decripta tale valore y e lo spedisce come firma al documento. Il destinatario usa invece la chiave pubblica $G*$ del mittente per ottenere m e verificare che corrisponda all'hash del documento; questo però non è possibile, perchè entrano in gioco le matrici causali S e P.
Nell'articolo già citato, viene proposto uno schema per la firma digitale basato sul sistema crittografico di Niederreiter, che è una variante di McEliece. Anche questo schema è però già stato rotto.

Capitolo 4

Codice di Fibonacci

4.1 Introduzione

In quest'ultimo capitolo riprenderemo parte del lavoro del matematico Stakhov, in particolare quanto scritto in Stakhov [2006]. L'autore parte dalla definizione classica di sequenza, o numeri, di Fibonacci: l'ennesimo numero di Fibonacci, indicato con $F(n)$, soddisfa la relazione ricorsiva

$$F(n) = F(n-1) + F(n-2) \tag{4.1}$$

per $n > 2$. I valori iniziali per $n = 1{,}2$ sono dati da:

$$F(1) = 1 \qquad F(2) = 1 \tag{4.2}$$

Talvolta è anche indicato il valore iniziale per $n = 0$, con $F(0) = 0$. I primi sei valori di $F(n)$ sono riportati in tabella 4.1

n	1	2	3	4	5	6
F(n)	1	1	2	3	5	8

Tabella 4.1: Primi 6 valori della successione di Fibonacci

Può essere utile, specialmente per valori grandi di n, avere una formula che permetta di calcolare $F(n)$ senza dover necessariamente conoscere $F(n-1)$ e $F(n-2)$. Possiamo per esempio cercare $F(n)$ nella forma x^n, dove in generale $x \in \mathbb{R}$. Osserviamo che questo implica

$$F(n-1) = x^{n-1}$$
$$F(n-2) = x^{n-2}$$

e che la ricorsione di (4.1) si può quindi riscrivere come $x^n = x^{n-1} + x^{n-2}$. Se portiamo tutto a sinistra e raccogliamo x^{n-2}, otteniamo l'equazione di secondo grado:

$$x^2 - x - 1 = 0 \tag{4.3}$$

dove il polinomio $p(x) = x^2 - x - 1$ è detto *polinomio caratteristico* della sucessione di Fibonacci. L'equazione (4.3) ha due soluzioni, che indichiamo con ψ e $\bar{\psi}$:

$$\psi = \frac{1 + \sqrt{5}}{2} \tag{4.4}$$

$$\bar{\psi} = \frac{1-\sqrt{5}}{2} \tag{4.5}$$

ψ e $\bar{\psi}$ sono quindi due valori che soddisfano la ricorsione di Fibonacci (4.1), cioè per cui si ha:

$$\psi^n = \psi^{n-1} + \psi^{n-2} \tag{4.6}$$
$$\bar{\psi}^n = \bar{\psi}^{n-1} + \bar{\psi}^{n-2} \tag{4.7}$$

Facciamo ora vedere che anche le combinazioni lineari di ψ e $\bar{\psi}$ sono una soluzione di (4.1). Siano $\alpha, \beta \in \mathbb{R}$

$$\alpha\psi^n + \beta\bar{\psi}^n =$$
$$\alpha(\psi^{n-1} + \psi^{n-2}) + \beta(\bar{\psi}^{n-1} + \bar{\psi}^{n-2}) = \alpha\psi^{n-1} + \beta\bar{\psi}^{n-1} + \alpha\psi^{n-2} + \beta\bar{\psi}^{n-2}$$

dove la prima uguaglianza segue dalle formule (4.6) e (4.7). La ricorsione di Fibonacci (4.1) sarà quindi soddisfatta da una combinazione lineare del tipo:

$$F(n) = \alpha\left(\frac{1+\sqrt{5}}{2}\right)^n + \beta\left(\frac{1-\sqrt{5}}{2}\right)^n$$

Imponendo i valori iniziali di (4.2), possiamo ricavare i valori di α e β:

$$\begin{cases} \alpha\left(\frac{1+\sqrt{5}}{2}\right) + \beta\left(\frac{1-\sqrt{5}}{2}\right) = 1 \\ \alpha\left(\frac{1+\sqrt{5}}{2}\right)^2 + \beta\left(\frac{1-\sqrt{5}}{2}\right)^2 = 1 \end{cases}$$

Il sistema ammette come soluzione:

$$\begin{cases} \alpha = \frac{1}{\sqrt{5}} \\ \beta = -\frac{1}{\sqrt{5}} \end{cases}$$

La successione di Fibonacci si può quindi scrivere come:

$$F(n) = \frac{1}{\sqrt{5}}(\psi^n - \bar{\psi}^n) \tag{4.8}$$

Questa formulazione è nota come *formula di Binet* Cotignoli [2018]; il valore ψ definito da (4.4) è invece chiamato *sezione aurea*. La successione di Fibonacci soddisfa la relazione:

$$\lim_{n\to\infty} \frac{F(n+1)}{F(n)} = \psi \tag{4.9}$$

Infatti, usando la formula di Binet (4.8):

$$\lim_{n\to\infty} \frac{F(n+1)}{F(n)} = \lim_{n\to\infty} \frac{\psi^{n+1} - \bar{\psi}^{n+1}}{\psi^n - \bar{\psi}^n} = \lim_{n\to\infty} \frac{\psi^n(\psi - \frac{\bar{\psi}^{n+1}}{\psi^n})}{\psi^n(1 - \frac{\bar{\psi}^n}{\psi^n})} = \lim_{n\to\infty} \frac{\psi - \bar{\psi}(\frac{\bar{\psi}}{\psi})^n}{1 - (\frac{\bar{\psi}}{\psi})^n} = \psi$$

dove l'ultima uguaglianza segue dal fatto che $\left|\frac{\bar{\psi}}{\psi}\right| < 1$.
Notiamo inoltre che, fissati i valori iniziali $F(1) = 1$ e $F(2) = 2$, possiamo scrivere:

$$\begin{aligned} F(3) &= F(2) + F(1) = F(1) + 1 \\ F(4) &= F(3) + F(2) = F(1) + F(2) + 1 \\ F(5) &= F(4) + F(3) = F(3) + F(2) + \underbrace{F(2)}_{=1} + F(1) = F(1) + F(2) + F(3) + 1 \end{aligned}$$

e così via. In generale si ha:

$$F(n+2) = F(1) + F(2) + \cdots + F(n) + 1 \tag{4.10}$$

Può anche essere utile vedere l'estensione ai numeri n negativi, definita da:

$$F(-n) = \begin{cases} -F(n) & \text{se } n \text{ pari} \\ F(n) & \text{se } n \text{ dispari} \end{cases} \tag{4.11}$$

Fatte queste premesse, possiamo procedere con i passaggi che ci porteranno, sulla falsa riga di quanto fatto da Stakhov, a definire il **codice di Fibonacci** Tranne dove espressamente indicato, il materiale di questo capitolo è tratto dai due articoli già citati.

4.2 Matrice Q di Fibonacci

Definizione 16. *Si definisce matrice Q di Fibonacci quella matrice quadrata* 2×2 *data da:*

$$Q = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

La matrice Q soddisfa la seguente proprietà:

Proprietà 10.

$$Q^n = \begin{pmatrix} F(n+1) & F(n) \\ F(n) & F(n-1) \end{pmatrix} \tag{4.12}$$

dove $F(n+1)$, $F(n)$ *e* $F(n-1)$ *indicano i rispettivi numeri di Fibonacci.*

Dimostrazione. Dimostriamo la proprietà 10 in modo induttivo. Sia quindi $n = 1$; è immediato verificare che:

$$Q^1 = Q = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} F(2) & F(1) \\ F(1) & F(0) \end{pmatrix}$$

Supponiamo ora che l'equazione (4.12) sia valida per n. Facciamo vedere che vale anche per $n+1$

$$Q^{n+1} = Q^n Q = \begin{pmatrix} F(n+1) & F(n) \\ F(n) & F(n-1) \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} F(n+1)+F(n) & F(n+1) \\ F(n)+F(n-1) & F(n) \end{pmatrix} = \begin{pmatrix} F(n+2) & F(n+1) \\ F(n+1) & F(n) \end{pmatrix}$$

Questo dimostra che Q^{n+1} soddisfa l'equazione (4.12). □

Vediamo ancora una proprietà sul determinante di Q^n, che useremo poi per dimostrare l'identità di Cassini.

Proprietà 11. *Il determinante di Q^n soddisfa la seguente relazione*

$$det(Q^n) = (-1)^n \tag{4.13}$$

Dimostrazione. Dalla formula (4.12), calcoliamo il determinante di Q^n:

$$\det(Q^n) = F(n+1)F(n-1) - F^2(n) \tag{4.14}$$

Sostituiamo le espressioni di $F(n+1)$, $F(n-1)$ e $F^2(n)$ con la formula di Binet (4.8):

$$\begin{aligned}
\det(Q^n) &= \frac{1}{\sqrt{5}}(\psi^{n+1} - \bar{\psi}^{n+1})\frac{1}{\sqrt{5}}(\psi^{n-1} - \bar{\psi}^{n-1}) - \frac{1}{5}(\psi^n - \bar{\psi}^n)^2 \\
&= -\frac{1}{5}(\psi\bar{\psi})^{n-1}(\psi - \bar{\psi})^2 \\
&= -\frac{1}{5}\left[\left(\frac{1+\sqrt{5}}{2}\right)\left(\frac{1-\sqrt{5}}{2}\right)\right]^{n-1}\left(\frac{1+\sqrt{5}}{2} - \frac{1-\sqrt{5}}{2}\right)^2 \\
&- \frac{1}{5}(-1)^{n-1}\cdot 5 = (-1)^n
\end{aligned}$$

□

Dalla proprietà 11, usando la formula 4.14, ricaviamo l'espressione nota come *identità di Cassini*:

$$F(n+1)F(n-1) - F^2(n) = (-1)^n \tag{4.15}$$

Facciamo vedere anche una formula che ci dà un'ulteriore analogia tra le Q e i numeri di Fibonacci:

$$\begin{aligned}
Q^n = \begin{pmatrix} F(n+1) & F(n) \\ F(n) & F(n-1) \end{pmatrix} &= \begin{pmatrix} F(n)+F(n-1) & F(n-1)+F(n-2) \\ F(n-1)+F(n-2) & F(n-2)+F(n-3) \end{pmatrix} \\
\begin{pmatrix} F(n) & F(n-1) \\ F(n-1) & F(n-2) \end{pmatrix} + \begin{pmatrix} F(n-1) & F(n-2) \\ F(n-2) & F(n-3) \end{pmatrix} &= Q^{n-1} + Q^{n-2}
\end{aligned}$$

Vediamo anche l'inverso della matrice Q, ma prima di farlo dimostriamo una proprietà utile per cacolare l'inversa di una matrice 2×2

Proprietà 12. *Sia A una matrice quadrata invertibile di dimensione 2×2:*

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$$

La matrice inversa di A, che indichiamo con A^{-1}, è data da:

$$A^{-1} = \frac{1}{\det A}\begin{pmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{pmatrix} \tag{4.16}$$

con $\det A = a_{11}a_{22} - a_{21}a_{12}$.

Dimostrazione. La dimostrazione è immediata: basta far vedere che

$$AA^{-1} = I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

.

$$AA^{-1} = \frac{1}{\det A}\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}\begin{pmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{pmatrix} = \frac{1}{\det A}\begin{pmatrix} a_{11}a_{22} - a_{21}a_{12} & 0 \\ 0 & a_{11}a_{22} - a_{21}a_{12} \end{pmatrix} = I$$

□

Usando quindi la formula (4.16) per Q^n, troviamo:

- se n è pari ($n = 2k$):

$$Q^{-2k} = \begin{pmatrix} F(2k-1) & -F(2k) \\ -F(2k) & F(2k+1) \end{pmatrix} \tag{4.17}$$

- invece se n è dispari ($n = 2k+1$):

$$Q^{-(2k+1)} = \begin{pmatrix} -F(2k) & F(2k+1) \\ F(2k+1) & -F(2k+2) \end{pmatrix} \tag{4.18}$$

Queste relazioni le useremo nella sezione 4.4 per le matrici Q_p generalizzate.

4.3 p numeri di Fibonacci

I p numeri di Fibonacci rappresentano una generalizzazione della classica sequenza di Fibonacci e possono essere così definiti

Definizione 17. *Sia $p > 0$ un intero fissato. I p numeri di Fibonacci, indicati con $F_p(n)$, soddisfano la seguente relazione ricorsiva:*

$$F_p(n) = F_p(n-1) + F_p(n-p-1) \qquad se \qquad n > p+1 \tag{4.19}$$

I valori iniziali sono dati da:

$$F_p(1) = F_p(2) = \cdots = F_p(p+1) = 1 \tag{4.20}$$

Osserviamo che, per $p = 0$, si ha:

- $F_0(1) = 1$ come valore iniziale
- $F_0(n) = F_0(n-1) + F_0(n-1) = 2F_0(n-1)$

La successione generata è quindi 0,1,2,...; in generale, $F_0(n) = 2^{n-1}$.
Invece per $p = 1$ abbiamo $F_1(n) = F_1(n-1) + F_2(n-2)$, che è la successione di Fibonacci classica che già conosciamo.
Anche per i numeri F_p si può definire l'estensione ai casi per $p \leq 0$. Infatti, per $n = p+1$, la formula (4.19) diventa:

$$F_p(p+1) = F_p(p) + F_p(0)$$

dove abbiamo già visto che $F_p(p+1) = F_p(p) = 1$, da cui $F_p(0) = 0$.
Continuando il ragionamento, scriviamo l'espressione di $F_p(p), F_p(p-1), \cdots F_p(2)$ troviamo:

$$F_p(0) = F_p(-1) = F_p(-2) = \cdots = F_p(-p+1) = 0$$

Per ottenere la rappresentazione di $F_p(-p)$, scriviamo la formula (4.19) per $F_p(0)$:

$$F_p(1) = F_p(0) + F_p(-p)$$

Siccome $F_p(1) = 1$ e $F_p(0) = 0$, allora $F_p(-p) = 1$. Come abbiamo fatto precedentemente, se scriviamo la stessa relazione per $F_p(0), F_p(-1), \cdots F_p(-p+1)$, troviamo:

$$F_p(-p-1) = F(-p-2) = \cdots F_p(-2p+1) = 0$$

e iterando questo processo otteniamo i valori per gli n negativi. Nella tabella 4.2 sono riportati alcuni esempi per $p = 1,2,3$.
Facciamo anche vedere che per p numeri di Fibonacci vale una relazione analoga alla (4.10).

n	5	4	3	2	1	0	-1	-2	-3	-4	-5
$F_1(n)$	5	3	2	2	1	0	1	-1	2	-3	5
$F_2(n)$	3	2	1	1	1	0	0	1	0	-1	1
$F_3(n)$	2	1	1	1	1	0	0	0	1	0	0

Tabella 4.2: p numeri di Fibonacci

Proprietà 13.

$$F_p(n+p+1) = F_p(1) + F_p(2) + \cdots + F_p(n) + 1 \tag{4.21}$$

Dimostrazione. Facciamo vedere per induzione sul valore di n. Sia $n = 1$. Si ha:

$$F_p(p+2) = F_p(1) + 1 = 2$$
$$F_p(p+2) = F_p(p+1) + F_p(1) = 1 + F_p(1) = 2$$

I due valori coincidono. Supponiamo adesso che l'equazione (4.21) sia valida fino a n e facciamo vedere che è valida anche per $n+1$.

$$F_p(n+p+2) = F_p(n+p+1) + F_p(n+1) = F_p(1) + F_p(2) + \cdots + F_p(n) + 1 + F_p(n+1).$$

□

4.4 Matrici di Fibonacci generalizzate

Una generalizzazione analoga a quella da $F(n)$ a $F_p(n)$, si può fare con le matrici Q. Definiamo quindi la matrice di Fibonacci generalizzata Q_p come la matrice quadarata di dimensione $(p+1) \times (p+1)$ come:

$$Q_p = \begin{pmatrix} 1 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \\ 0 & 0 & 0 & 0 & \cdots & 1 \\ 1 & 0 & 0 & 0 & \cdots & 0 \end{pmatrix} \tag{4.22}$$

Si tratta di una matrice identità I_p contornata da una colonna e da una riga del tipo $1,0,\cdots,0$. Vediamo qualche esempio di Q_p:

$$Q_1 = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} = Q \qquad Q_2 = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

$$Q_3 = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix} \qquad Q_4 = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Il determinante della matrice Q_p soddisfa il seguente teorema

Teorema 14.

$$\det Q_p = (-1)^p \tag{4.23}$$

Dimostrazione. Usiamo la regola di Laplace per calcolare qualche determinante di Q_p:

$$\det Q_1 = -1$$

$$\det Q_2 = \det \begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} = 1 \cdot (-1)^5 \det \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} = -1 \det Q_1 = 1$$

$$\det Q_3 = \det \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix} = (-1)^7 \det Q_2 = -1$$

Otteniamo in questo modo anche una formula ricorsiva per il calcolo del determinante di Q_p:

$$\det Q_p = 1\dot{(}-1)^{2p+1} \det Q_{p-1} = (-1)\dot{Q}_{p-1} \tag{4.24}$$

perchè $2p+1$ è dispari. Vediamo con alcuni valori di p:

- per $p = 2$ abbiamo $Q_2 = (-1) \det Q_1 = (-1)^2$
- per $p = 3$ abbiamo $Q_3 = (-1) \det Q_2 = (-1)^3$
- per $p = 4$ abbiamo $Q_4 = (-1) \det Q_3 = (-1)^4$

e così via, arrivando ad avere, in generale, $\det Q_p = (-1)^p$. □

4.4.1 Elevamento a potenza delle matrici di Fibonacci

Anche per l'elevamento a potenza di Q_p vale una formula analoga a Q

Teorema 15. *Sia p un intero fissato. La matrice Q_p^n è data da:*

$$Q_p^n = \begin{pmatrix} F_p(n+1) & F_p(n) & \dots & F_p(n-p+2) & F_p(n-p+1) \\ F_p(n-p+1) & F_p(n-p) & \cdots & F_p(n-2p+2) & F_p(n-2p+1) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ F_p(n-1) & F_p(n-2) & \dots & F_p(n-p) & F_p(n-p-1) \\ F_p(n) & F_p(n-1) & \vdots & F_p(n-p+1) & F_p(n-p) \end{pmatrix} \tag{4.25}$$

Omettiamo la dimostrazione, che si fa per induzione similmente a quello fatto per la proprietà 10 nel caso della matrice Q. Vediamo qualche esempio.

Esempio 31. *Sia $p = 2$. La matrice Q_2^n è data da:*

$$Q_2^n = \begin{pmatrix} F_2(n+1) & F_2(n) & F_2(n-1) \\ F_2(n-1) & F_2(n-2) & F_2(n-3) \\ F_2(n) & F_2(n-1) & F_2(n-2) \end{pmatrix}$$

Se n=2, abbiamo:

$$Q_2^2 = \begin{pmatrix} F_2(3) & F_2(2) & F_2(1) \\ F_2(1) & F_2(0) & F_2(-1) \\ F_2(2) & F_2(1) & F_2(0) \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{pmatrix}$$

Esempio 32. *Invece per $p = 3$ troviamo:*

$$Q_3^n = \begin{pmatrix} F_3(n+1) & F_3(n) & F_3(n-1) & F_3(n-2) \\ F_3(n-2) & F_3(n-3) & F_3(n-4) & F_3(n-5) \\ F_3(n-1) & F_3(n-2) & F_3(n-3) & F_3(n-4) \\ F_3(n) & F_3(n-1) & F_3(n-2) & F_3(n-3) \end{pmatrix}$$

Prendiamo anche questa volta il caso di $n = p = 3$:

$$Q_3^3 = \left(\begin{array}{ccc|c} 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ \hline 1 & 1 & 1 & 0 \end{array}\right)$$

Questa scrittura ci fa notare come per ottenere Q_3^3 basti prendere Q_2^2 e aggiungervi come ultima colonna il vettore 1,0,0,0 e come ultima riga 1,1,1,0.

Il determinante di Q_p^n soddisfa la seguente proprietà:

Proprietà 14.

$$\det Q_p^n = (-1)^{pn} \tag{4.26}$$

Dimostrazione. La dimostrazione è immediata e segue immediatamente dalla formula (4.13) per il determinante di Q^n. □

La formula (4.26) è detta *formula di Cassini generalizzata.* Tale formula si presenta, nel caso di $p = 2$, come:

$$\begin{aligned} \det Q_2^n = {} & F_2(n+1)\left[F_2^2(n-2) - F_2(n-1)F_2(n-3)\right] \\ & - F_2(n)\left[F_2(n-1)F_2(n-2) - F_2(n)F_2(n-3)\right] \\ & + F_2(n-1)\left[F_2^2(n-1) - F_2(n)F_2(n-2)\right] = (-1)^{2n} = 1 \end{aligned}$$

4.5 Codice di Fibonacci

Nel suo articolo Stakhov spiega come possiamo basarci sulle matrici Q_p^n per costruire un codice correttore in questo modo:

- per la codifica, dividiamo il messaggio in matrici M di dimensione $(p+1) \times (p+1)$ per p fissato e moltiplichiamo per Q_p^n, ottenendo così la matrice del codice C:

$$M \cdot Q_p^n = C \tag{4.27}$$

- per la decodifica, moltiplichamo la matrice C per Q_p^{-n} e otteniamo così la matrice M' che, se non ci sono stati errori, coincide con M:

$$C \cdot Q_p^{-n} = M' \tag{4.28}$$

Queste due operazioni vengono chiamate, rispettivamente, *codifica* e *decodifica di Fibonacci.*

Esempio 33. *Supponiamo di codificare un messaggio M di dimensione 2×2 del tipo:*

$$M = \begin{pmatrix} m_1 & m_2 \\ m_3 & m_4 \end{pmatrix}$$

Scegliamo di usare la matrice Q^6 per codificare:

$$Q^6 = \begin{pmatrix} 13 & 8 \\ 8 & 5 \end{pmatrix}$$

Per decodificare useremo quindi Q^{-6}:

$$Q^{-6} = \begin{pmatrix} 5 & -8 \\ -8 & 13 \end{pmatrix}$$

Codifica:

$$M \cdot Q^6 = \begin{pmatrix} m_1 & m_2 \\ m_3 & m_4 \end{pmatrix} \begin{pmatrix} 13 & 8 \\ 8 & 5 \end{pmatrix} = \begin{pmatrix} 13m_1 + 8m_2 & 8m_1 + 5m_2 \\ 13m_3 + 8m_4 & 8m_3 + 5m_4 \end{pmatrix} = \begin{pmatrix} c_1 & c_2 \\ c_3 & c_4 \end{pmatrix} = C$$

Decodifica:

$$C \cdot Q^{-6} = \begin{pmatrix} c_1 & c_2 \\ c_3 & c_4 \end{pmatrix} \begin{pmatrix} 5 & -8 \\ -8 & 13 \end{pmatrix} = \begin{pmatrix} 13m_1 + 8m_2 & 8m_1 + 5m_2 \\ 13m_3 + 8m_4 & 8m_3 + 5m_4 \end{pmatrix} \begin{pmatrix} 5 & -8 \\ -8 & 13 \end{pmatrix} = \begin{pmatrix} m_1' & m_2' \\ m_3' & m_4' \end{pmatrix} = M'$$

Osserviamo che quest'equazione ha soluzione:

$$\begin{cases} m_1' = m_1 \\ m_2' = m_2 \\ m_3' = m_3 \\ m_4' = m_4 \end{cases}$$

Osservazione 6. *Il determinante di $C = M \cdot Q_p^n$ sarà in generale dato da:*

$$\det C = \det M \cdot \det Q_p^n = \det M (-1)^{pn} \tag{4.29}$$

dove l'ultima uguaglianza segue dalla (4.26). Quindi il determinante di C può al massimo differire per il segno rispetto al determinante di M.

4.5.1 Relazioni tra gli elementi della matrice C

Sia

$$M = \begin{pmatrix} m_1 & m_2 \\ m_3 & m_4 \end{pmatrix}$$

il messaggio da codificare. Non è restrittivo supporre che

$$m_1, m_2, m_3, m_4 > 0 \tag{4.30}$$

in quanto, se non lo sono, è sufficiente sommare ad M una matrice A appropriata. Sia ora

$$C = \begin{pmatrix} c_1 & c_2 \\ c_3 & c_4 \end{pmatrix}$$

la matrice ottenuta dalla codifica di Fibonacci come $C = M \cdot Q^n$. Se decodifichiamo C, supponendo di non avere errori, troviamo:

$$M = \begin{pmatrix} m_1 & m_2 \\ m_3 & m_4 \end{pmatrix} = \begin{pmatrix} c_1 & c_2 \\ c_3 & c_4 \end{pmatrix} \cdot \begin{pmatrix} -F(n-1) & F(n) \\ F(n) & -F(n+1) \end{pmatrix}$$

se prendiamo n dispari. Otteniamo quindi il sistema:

$$\begin{cases} m_1 = -F(n-1)c_1 + F(n)c_2 \\ m_2 = F(n)c_1 - F(n+1)c_2 \\ m_3 = -F(n-1)c_3 + F(n)c_4 \\ m_4 = F(n)c_3 - F(n+1)c_4 \end{cases}$$

Se imponiamo le condizioni (4.30), abbiamo:

$$\begin{cases} -F(n-1)c_1 + F(n)c_2 > 0 \\ F(n)c_1 - F(n+1)c_2 > 0 \\ -F(n-1)c_3 + F(n)c_4 > 0 \\ F(n)c_3 - F(n+1)c_4 > 0 \end{cases}$$

da cui ricaviamo:

$$\begin{aligned} \frac{F(n+1)}{F(n)}c_2 < c_1 < \frac{F(n)}{F(n-1)}c_2 \\ \frac{F(n+1)}{F(n)}c_4 < c_3 < \frac{F(n)}{F(n-1)}c_4 \end{aligned}$$

che possiamo riscrivere come

$$\begin{aligned} \frac{F(n+1)}{F(n)} < \frac{c_1}{c_2} < \frac{F(n)}{F(n-1)} \\ \frac{F(n+1)}{F(n)} < \frac{c_3}{c_4} < \frac{F(n)}{F(n-1)} \end{aligned}$$

Ma poichè vale (4.9)allora deve essere:

$$c_1 \approx \psi c_2 \tag{4.31}$$
$$c_3 \approx \psi c_4 \tag{4.32}$$

dove ψ è la sezione aurea: $\psi = \frac{1+\sqrt{5}}{2}$. Vale un ragionamento analogo nel caso di n dispari.

4.5.2 Riconoscimento e correzione di errori

Per riconoscere gli errori possiamo usare i due determinanti $\det C$ e $\det M$: se infatti tali quantità soddisfano la relazione (4.29), allora possiamo concludere che non c'è stato alcun errore; viceversa, se tale relazione non è valida, allora siamo in presenza di un errore. Per questo motivo, Stakhov suggerisce di inviare, oltre alla matrice di codice C, anche il determinante di M come elemento di controllo (*checking element*).

Errore singolo

Ipotizziamo per prima cosa di avere un errore singolo nella matrice C. I casi possibili con C di dimensione 2×2 sono:

$$\begin{pmatrix} x & c_2 \\ c_3 & c_4 \end{pmatrix} \qquad \begin{pmatrix} c_1 & y \\ c_3 & c_4 \end{pmatrix} \qquad \begin{pmatrix} c_1 & c_2 \\ z & c_4 \end{pmatrix} \qquad \begin{pmatrix} c_1 & c_2 \\ c_3 & t \end{pmatrix} \tag{4.33}$$

Per capire in quale posizione all'interno della matrice C sia presente l'errore, usiamo il fatto che $\det C = (-1)^n \det M$, esplicitando il determinante di C per i quattro casi:

$$\begin{aligned} xc_4 - c_2c_3 &= (-1)^n \det M \qquad \text{errore in prima posizione} \\ c_1c_4 - yc_3 &= (-1)^n \det M \qquad \text{errore in seconda posizione} \\ c_1c_4 - c_2z &= (-1)^n \det M \qquad \text{errore in terza posizione} \\ c_1t - c_2c_3 &= (-1)^n \det M \qquad \text{errore in quarta posizione} \end{aligned}$$

che diventano:

$$x = \frac{(-1)^n \det M + c_2c_3}{c_4} \tag{4.34}$$

$$y = \frac{-(-1)^n \det M + c_1c_4}{c_3} \tag{4.35}$$

$$z = \frac{-(-1)^n \det M + c_1c_4}{c_2} \tag{4.36}$$

$$t = \frac{(-1)^n \det M + c_2c_3}{c_1} \tag{4.37}$$

Dobbiamo cercare delle soluzioni intere che soddisfino le relazioni (4.31) e (4.32). Se queste soluzioni intere non esistono, allora significa che c'è stato o un errore nel determinante di M o che nella matrice C è presente più di un errore. Prima di procedere con il caso dell'errore doppio, vediamo un semplice esempio.

Esempio 34. *Supponiamo che la matrice del messaggio sia* $M = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$ *che ha* $\det M = 1$. *Per codificare usiamo la matrice* $Q^2 = \begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix}$ *La matrice* C *di codfica è data da:*

$$C = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 3 & 2 \\ 1 & 1 \end{pmatrix}$$

Osserviamo che $\det C = 3 - 2 = 1 = \det M(-1)^2$. *Ipotizziamo adesso che, durante la trasmissione, avvenga un errore nella prima posizione di* C *e la matrice ricevuta risulti essere*

$$C_E = \begin{pmatrix} 2 & 2 \\ 1 & 1 \end{pmatrix}$$

mentre $\det M = 1$ *viene trasmesso correttamente. Vediamo subito che si ha* $\det C_E = 2 - 2 = 0 \neq 1$, *perciò si è verificato un errore. Se usiamo la relazione* (4.34), *abbiamo:*

$$x = \frac{(-1)^2 \cdot 1 + 2}{1} = 3 \neq 2$$

quindi l'errore si trova nella prima componente di C_E. Per calcolare il valore corretto, sfuttiamo la relazione (4.31):

$$c_1 = \psi \cdot 2 = 1 + \sqrt{2} \approx 3$$

Osserviamo infatti che se mettiamo 3 in prima posizione, oteniamo la relazione tra i determinanti corretta.

Errore doppio e triplo

Consideriamo una matrice C_E con due errori del tipo:

$$C_E = \begin{pmatrix} x & y \\ c_3 & c_4 \end{pmatrix}$$

Per correggere questo tipo di errori usiamo la relazioni tra i determinanti (4.29) e la formula (4.31) per scrivere:

$$xc_4 - yc_3 = (-1)^n \det M \tag{4.38}$$

$$x \approx \psi y \tag{4.39}$$

Siccome l'equazione (4.38) è diofantina, ammette molteplici soluzioni; tra queste, scegliamo quelle che soddisfano anche (4.39).
Un ragionamento analogo si fa per matrici C_E avente due errori in altre posizioni.

Si può correggere anche un errore triplo del tipo

$$\begin{pmatrix} x & y \\ z & c_4 \end{pmatrix}$$

sempre seguendo lo stesso metodo, quindi usando le relazioni:

$$xc_4 - yz = (-1)^n \det M$$
$$x \approx \psi y$$
$$z \approx \psi t$$

In totale, usando il codice di Fibonacci si possono correggere 14 errori su 15 totali, con una percentuale di correttezza del 93,33%.

4.5.3 Confronto con altri codici

Il punto di forza del codice di Fibonacci è proprio quello che abbiamo visto, cioè la percentuale di correttezza, che è molto più alta rispetto a praticamente tutti gli altri codici conosciuti.
Il codice di Fibonacci non è però un codice lineare. Lo possiamo vedere se prendiamo ad esempio $p = 1$; l'operazione di codifica si può scrivere come

$$\begin{pmatrix} F(n+1) & F(n) & 0 & 0 \\ F(n) & F(n-1) & 0 & 0 \\ 0 & 0 & F(n+1) & F(n) \\ 0 & 0 & F(n) & F(n-1) \end{pmatrix} \cdot \begin{pmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \end{pmatrix} = \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{pmatrix}$$

dove la prima matrice non è però una matrice generatrice, in quanto non presenta nessun blocco del tipo I_k (matrice identità di ordine k). Inoltre, rispetto ad altri codici (lineari e non lineari) mancano le difinizioni di peso e di distanza.
Nela prossima sezione, vedremo i principali ostacoli per costruire un sistema crittografico basato sul codice di Fibonacci

4.6 Implementazione di un sistema crittografico di Fibonacci

In analogia con quanto fatto per il sistema di McEliece, analizziamo i principali step per costruire un sistema crittografico a chiave pubblica basato sui codici di Fibonacci.
Ciascun utente sceglie due interi p e n che usa per calcolare la matrice Q_p^n; sceglie poi anche due matrici casuali S e P (di permutazione), di dimensione $(p+1) \times (p+1)$, con l'unico vincolo che S deve essere invertibile. Queste matrici sono poi moltiplicate con Q_p^n per ottenere la matrice $Q* = SQ_p^nP$. La coppia $Q*, p$ è resa nota (*chiave pubblica*), mentre rimangono segreti i valori di S, n e Q_p^n, che formeranno quindi la *chiave privata*. Per cifrare, occorre:

- dividere il messaggio in blocchi matriciali M di dimensione $(p+1) \times (p+1)$
- moltiplicare ciascun blocco M per $Q*$
- aggiungere la matrice E di errore

Il messaggio cifrato sarà quindi la matrice $X = MQ* + E$, che viene poi spedita insieme al determinante di M.

Per decifrare, il destinatario:

- calcola la matrice $Y = XP^{-1} = MSQ_p^n + EP^{-1} = M'Q_p^n + E'$, dove si è posto $M' = MS$ e $E' = EP^{-1}$
- utilizza l'algoritmo di correzione degli errori per ricavare $C = M'Q_p^n$ (per fare questa operazione ha bisogno del determinante di M', che si calcola con la formula $\det(M') = \det M \det S$)
- moltiplica M' per S^{-1} per ottenere M

Tale sistema crittografico è descritto sinteticamente nella tabella 4.3.

chiave privata	n, Q_p^n, S, P
chiave pubblica	$Q* = SQ_p^nP$, p
cifratura	$X = MQ* + E$, con E matrice di errore
decifratura	calcolo $Y = XP^{-1} = M'Q_p^n + E'$ correggo errori e trovo $C = M'Q_p^n$ moltiplico per S^{-1} e trovo M

Tabella 4.3: Schematizzazione del sistema crittografico di Fibonacci

Prima di vedere un esempio, diamo un'idea di come si possa correggere l'errore nella fase di decodifica, provando a generalizzare quanto fatto da Stakhov nel caso di $p = 1$. Vediamo prima un paio di risultati ausiliari.

4.6.1 Foma chiusa e limite degli Fp

Sia $n = 0,1,2,\cdots$. L'idea è quella di trovare una relazione per l'andamento di $F_p(-n+k)$ per $-n \to -\infty$, in analogia con la relazione (4.9) per i numeri di Fibonacci classici. Sia $p > 1$ fissato. La sequenza di Fibonacci F_p è data da:

$$F_p(n) = F_p(n-1) + F_p(n-p-1)$$

con $F_p(1) = F_p(2) = \cdots = F_p(p+1) = 1$. Per trovare gli $F_p(-n)$, seguendo il ragionamento fatto nella sezione 4.3, scriviamo la relazione inversa:

$$F_p(n-p-1) = F_p(n) - F_p(n-1)$$

dove n deve essere tale che $n-p-1 < 0$. Ad esempio, se $n = p$, otteniamo $F_p(-1) = F_p(p) - F_p(p-1) = 0$, se $n = p-1$ $F(-2) = F_p(p-1) - F_p(p-2) = 0$ e così via. Per comodità, prendiamo $t = n-p-1$ e scriviamo

$$F_p(t) = F_p(t+p+1) - F_p(t+p) \tag{4.40}$$

con $t = -1, -2, \ldots$.
Per costruire il polinomio carattersitico, imponiamo $F_p(t) = x^t$, da cui $F_p(t+p+1) = x^{t+p+1}$ e $F_p(t+p) = x^{t+p}$. La formula (4.40) diventa quindi:

$$x^t = x^{t+p+1} - x^{t+p}$$

Se escludiamo la soluzione banale $x = 0$ e dividiamo a destra e a sinistra per x^t, otteniamo l'equazione $1 = x^{p+1} - x^p$, la cui risoluzione corrisponde al trovare gli zeri del polinomio caratteristico

$$f(x) = x^{p+1} - x^p - 1 \tag{4.41}$$

Per il teorema di esistenza degli zeri teorema Bolzano - Wikipedia, poichè risulta:

$$\begin{aligned} &f(1) = -1 < 0 \\ &f(2) = 2^{p+1} - 2^p - 1 = 2^p(2-1) - 1 = 2^p - 1 > 0 \qquad \text{per } p \geq 1 \end{aligned}$$

allora esiste un numero α nell'intervallo $I = (1,2)$ tale che $f(\alpha) = 0$. Inoltre possiamo avere due casi.

- Se p è **pari**, il polinomio caratteristico descritto da (4.41) ha grado dispari, pertanto è monotono crescente per $x < 0$ e per $x > \frac{p}{p+1}$ e monotono decrescente per $0 < x < \frac{p}{p+1}$; nell'intervallo $(-\infty,0]$ si ha $f(x) < 0$, da cui concludiamo che l'unico zero di $f(x)$ è α. In questo caso, $F_p(t) = \alpha^n$
- se p è **dispari**, $f(x)$ ha grado pari ed è monotono decrescente per $x < \frac{p}{p+1}$ e monotono crescente per $x > \frac{p}{p+1}$; applicando nuovamente il teorema di esistenza degli zeri sull'intervallo $[-1,0]$, vediamo che esiste $\beta \in (-1,0)$ tale che $f(\beta) = 0$. In questo caso, la ricorsione di Fibonacci sarà soddisfatta da una combinazione lineare di α e β: $F_p(t) = A\alpha^n + B\beta^n$, dove in generale $A, B \in \mathbb{R}$.

Per correggere gli errori possiamo usare il limite per $t \to -\infty$ di $\frac{F_p(t+k)}{F_p(t)}$ per qualche $k \in \mathbb{Z}$. Nel caso di p pari:

$$\lim_{t\to-\infty} \frac{F_p(t+k)}{F_p(t)} = \lim_{t\to-\infty} \frac{\alpha^{t+k}}{\alpha^t} = \alpha^k \tag{4.42}$$

mentre nel caso di p dispari:

$$\lim_{t\to-\infty} \frac{F_p(t+k)}{F_p(t)} = \lim_{t\to-\infty} \frac{A\alpha^{t+k} + B\beta^{t+k}}{A\alpha^t + B\beta^t} = \lim_{t\to-\infty} \left(\alpha^k \frac{A\alpha^t}{\alpha^t \left(A + B\left(\frac{\beta}{\alpha}\right)^t\right)} + \beta^k \frac{B\beta^t}{\beta^t \left(A\left(\frac{\alpha}{\beta}\right)^t + B\right)} \right) = \beta^k \tag{4.43}$$

Nella tabella 4.4 sono sintetizzati i risultati che abbiamo trovato:

	forma chiusa	$\lim_{-n\to-\infty} \frac{F_p(t+k)}{F_p(t)}$
p dispari	$F_p(t) = A\alpha^n + B\beta^n$, con $A, B \in \mathbb{R}$	β^k
p pari	$F_p(t) = \alpha^n$	α^k

Tabella 4.4: Sintesi dei risultati

4.6.2 Correzione di errori (idea)

Fissiamo $p > 1$ e n. Sia M la matrice del messaggio, che codifichiamo con il codice di Fibonacci per ottenere $C = MQ_p^n$. Se non si verificano errori, ricostruiamo M nella procedura di decodifica $M = CQ_p^n$. Nel dettaglio, avremo:

$$\begin{pmatrix} m_1 & m_2 & \dots & m_{p+1} \\ m_{p+2} & m_{p+3} & \dots & m_{2p+2} \\ m_{2p+3} & m_{2p+4} & \dots & m_{3p+3} \\ \vdots & \vdots & \vdots & \vdots \\ m_{p^2} & m_{p^2+1} & \dots & m_{p^2+p} \\ m_{p^2+p+1} & m_{p^2+p+2} & \dots & m_{p^2+2p+1} \end{pmatrix} =$$

$$\begin{pmatrix} c_1 & c_2 & \dots & c_{p+1} \\ c_{p+2} & c_{p+3} & \dots & c_{2p+2} \\ c_{2p+3} & c_{2p+4} & \dots & c_{3p+3} \\ \vdots & \vdots & \vdots & \vdots \\ c_{p^2} & c_{p^2+1} & \dots & c_{p^2+p} \\ c_{p^2+p+1} & c_{p^2+p+2} & \dots & c_{p^2+2p+1} \end{pmatrix} \begin{pmatrix} F_p(-n+1) & F_p(-n) & \dots & F_p(-n-p+1) \\ F_p(-n-p+1) & F_p(-n-p) & \dots & F_p(-n-2p+1) \\ \vdots & \vdots & \vdots & \vdots \\ F_p(-n-1) & F_p(-n-2) & \dots & F_p(-n-p-1) \\ F_p(-n) & F_p(-n-1) & \dots & F_p(-n-p) \end{pmatrix}$$

Come già aveva fatto Stakhov, non è restrittivo supporre gli $m_i > 0\ \forall i$, da cui possiamo scrivere il seguente sistema di disequazioni:

$$\begin{cases} c_1F_p(-n+1) + c_2F_p(-n-p+1) + \dots + c_pF_p(-n-1) + c_{p+1}F_p(-n) > 0 \\ c_1F_p(-n) + c_2F_p(-n-p) + \dots + c_pF_p(-n-2) + c_{p+1}F_p(-n-1) > 0 \\ \vdots \\ c_1F_p(-n-p+1) + c_2F_p(-n-2p+1) + \dots + c_pF_p(-n-p-1) + c_{p+1}F_p(-n-p) > 0 \end{cases}$$

e altri sistemi analoghi si possono scrivere per le righe $2,3,\cdots,p+1$, ciascuno con $p+1$ disequazioni. Isolando a sinistra c_1:

$$\begin{cases} c_1 > -\dfrac{1}{F_p(-n+1)} \left(\displaystyle\sum_{i=1}^{p-2} F_p(-n-ip+1)c_{i+1} + \sum_{i=-1}^{0} F_p(-n+i)c_{p+i+1} \right) \\ c_1 > -\dfrac{1}{F_p(-n)} \left(\displaystyle\sum_{i=1}^{p-2} F_p(-n-ip)c_{i+1} + \sum_{i=-1}^{0} F_p(-n+i-1)c_{p+i+1} \right) \\ \vdots \\ c_1 > -\dfrac{1}{F_p(-n-p+1)} \left(\displaystyle\sum_{i=1}^{p-2} F_p(-n-(i+1)p+1)c_{i+1} + \sum_{i=-1}^{0} F_p(-n-p+i)c_{p+i+1} \right) \end{cases}$$

Supponiamo p pari e facciamo tendere $-n \to -\infty$:

$$\begin{cases} c_1 > -\dfrac{1}{\alpha^{-n+1}}\left(\displaystyle\sum_{i=1}^{p-2}\alpha^{-n-ip+1}c_{i+1} + \sum_{i=-1}^{0}\alpha^{-n+i}c_{p+i+1}\right) = -\left(\displaystyle\sum_{i=1}^{p-2}\alpha^{-ip}c_{i+1} + \sum_{i=-1}^{0}\alpha^{i-1}c_{p+i+1}\right) \\ c_1 > -\dfrac{1}{\alpha^{-n}}\left(\displaystyle\sum_{i=1}^{p-2}\alpha^{-n-ip}c_{i+1} + \sum_{i=-1}^{0}\alpha^{-n+i-1}c_{p+i+1}\right) = -\left(\displaystyle\sum_{i=1}^{p-2}\alpha^{-ip}c_{i+1} + \sum_{i=-1}^{0}\alpha^{i-1}c_{p+i+1}\right) \\ \vdots \\ c_1 > -\dfrac{1}{\alpha^{-n-p+1}}\left(\displaystyle\sum_{i=1}^{p-2}\alpha^{-n-(i+1)p+1}c_{i+1} + \sum_{i=-1}^{0}\alpha^{(-n-p+i)}c_{p+i+1}\right) = -\left(\displaystyle\sum_{i=1}^{p-2}\alpha^{-ip}c_{i+1} + \sum_{i=-1}^{0}\alpha^{i-1}c_{p+i+1}\right) \end{cases}$$

Sommando queste disequazioni (che sono in totale $p+1$) abbiamo:

$$c_1 > -\frac{1}{p+1}\left((p+1)\sum_{i=1}^{p-2}\alpha^{-ip}c_{i+1} + (p+1)\sum_{i=-1}^{0}\alpha^{i-1}c_{p+i+1}\right) = -\left(\sum_{i=1}^{p-2}\alpha^{-ip}c_{i+1} + \alpha^{-2}c_p + \alpha^{-1}c_{p+1}\right) \tag{4.44}$$

Si possono scrivere delle relazioni analoghe per $c_{p+2}, c_{2p+3}, \ldots$. Ad esempio, nel caso di $p = 2$, le disequzioni diventano:

$$\begin{cases} c_1 > -\left(\alpha^{-2}c_2 + \alpha^{-1}c_3\right) \\ c_4 > -\left(\alpha^{-2}c_5 + \alpha^{-1}c_6\right) \\ c_7 > -\left(\alpha^{-2}c_8 + \alpha^{-1}c_9\right) \end{cases}$$

Facendo dei conti analoghi, nel caso di p dispari avremo la relazione:

$$c_1 > -\left(\sum_{i=1}^{p-2}\beta^{-ip}c_{i+1} + \beta^{-2}c_p + \beta^{-1}c_{p+1}\right) \tag{4.45}$$

Si intuisce facilmente che queste relazioni non sono però sufficienti per correggere gli errori. Mostriamo un caso in cui questa correzione degli errori non è efficace, e il sistema crittografico di Fibonacci, per come lo abbiamo impostato, fallisce.

Esempio 35. *Sia $p = 2$. Supponiamo che l' utente j abbia:*

- *come **chiave privata** le matrici*

$$Q_2^7 = \begin{pmatrix} 9 & 6 & 4 \\ 4 & 3 & 2 \\ 6 & 4 & 3 \end{pmatrix} \qquad S = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} \qquad P = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

- *come **chiave pubblica** $Q*$ il loro prodotto:*

$$Q* = \begin{pmatrix} 6 & 13 & 9 \\ 9 & 19 & 13 \\ 7 & 15 & 10 \end{pmatrix}$$

Se l'utente i vuole mandare a j il seguente messaggio

$$M = \begin{pmatrix} 7 & 0 & 12 \\ 1 & 20 & 17 \\ 6 & 4 & 17 \end{pmatrix}$$

lo cifra moltiplicandolo per $Q*$ e sommandolo con una matrice di errore E. Dimostriamo che, anche solo per una matrice E del tipo

$$E = \begin{pmatrix} 0 & e_2 & 0 \\ 0 & e_5 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

le condizioni che abbiamo descritto non sono sufficienti per corregere l'errore, indipendentemente dai valori di e_2 ed e_6.
Il messaggio cifrato sarà quindi:

$$X = \begin{pmatrix} 126 & 271 + e_2 & 183 \\ 305 & 648 + e_5 & 439 \\ 191 & 409 & 276 \end{pmatrix}$$

che viene spedito inisieme al determinante di M: $\det M = 512$. L'utente j calcola la matrice Y moltiplicando per P^{-1}:

$$Y = XP^{-1} = \begin{pmatrix} 271 + e_2 & 183 & 126 \\ 648 + e_5 & 439 & 305 \\ 409 & 276 & 191 \end{pmatrix}$$

Per correggere gli errori, usiamo la formula:

$$\det Y = \det M' = \det M \det S = 512 \tag{4.46}$$

Se poniamo $x = 271 + e_2$ e $y = 648 + e_5$, l'equazione (4.46) diventa

$$331x + 177y = 204.397$$

che ha come soluzione la coppia

$$x = -15.738.569$$

$$y = 29.433.168$$

che è stata calcolata usando Online calculator, in cui però la x non soddisfa la relazione

$$x > -(\alpha^{-2}183 + \alpha^{-1}126) > -309$$

Dobbiamo pertanto colcludere che in questo caso non siamo in grado di correggere l'errore.

In conclusione, possiamo affermare che la strada per costruire un ipotetico sistema critografico di Fibonacci, sul solco della crittografia post quantum e in particolare del lavoro fatto da McEliece, appare ben tracciata, ma restano alcuni problemi di non facile soluzione, primo fra tutti quello della correzione degli errori. In questo senso un ostacolo ulteriore è legato alla permutazione della matrice E. Si rimanda a lavori futuri lo studio di tale matrice e il suo impatto sul processo di decifratura. Potrebbe inoltre anche essere utile avere una nozione di peso e/o di distanza, in modo da capire fino a che limite ci si può spingere con i valori degli errori affinche questi risultino correggibili.

Bibliografia

Margherita Barile. Lezione 6 del corso di algebra 1. http://www.dm.uniba.it/~barile/Rete4/algebra1_pdf/lezione6.pdf, 2018. url consultata il 17-04-2019.

Dan Boneh. Twenty years of attacks on the rsa cryptosystem. *Notices of the AMS*, pages 203–213, 1999.

britannica. Cryptography-britannica.com. https://www.britannica.com/topic/cryptography, 2009. url consultata il 02-05-2019.

Paolo Caressa. Cos'è la crittografia? http://www.caressa.it/pdf/crypto.pdf, 2005. url consultata il 16-04-2019.

Cauchy's theorem - Wikipedia. Cauchy's theorem (group theory) - wikipedia. https://en.wikipedia.org/wiki/Cauchy%27s_theorem_(group_theory). url consultata il 18-05-2019.

characteristic of a field. Characteristic of a field is 0 or prime [closed]. https://math.stackexchange.com/questions/86263/characteristic-of-a-field-is-0-or-prime, 2011. url consultata il 06-06-2019.

characteristic of a field is a prime. Characteristic of a field f is prime [duplicate]. https://math.stackexchange.com/questions/573170/characteristic-of-a-field-f-is-prime, 2013. url consultata il 06-06-2019.

Lily Chen, Stephen Jordan, Yi-Kai Liu, Dustin Moody, Rene Peralta, Ray Perlner, and Daniel Smith-Tone. Nistir 8105 report on post-quantum cryptography. https://nvlpubs.nist.gov/nistpubs/ir/2016/NIST.IR.8105.pdf, 2016. url consultata il 06-06-2019.

code theory. Coding theory. https://www.mat.unical.it/~vanbon/Crypto/codes.pdf. url consultata il 20-05-2019.

Silvia Cotignoli. On a coding theory based on fibonacci numbers and linear recurrent sequences. Master's thesis, Università degli studi di Torino, 2018.

Nicolas T. Courtois, Matthieu Finiasz, and Nicolas Sendrier. How to achieve a mceliece-based digital signature scheme. In Colin Boyd, editor, *Advances in Cryptology — ASIACRYPT 2001*, pages 157–174, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

Charles C.Pinter. *A book of Abstract Algebra*. Dover publications, 2010.

Susan Curtis. Google aims for quantum supremacy. https://physicsworld.com/a/google-aims-for-quantum-supremacy/, 2018. url consultata il 02-06-2019.

Diffie and Hellman. New directions in cryptography. *IEEE Transactions on Information Theory.*, 22(6):644–654, 1976.

Discrete Root - algorithms. Discrete root-competitve programming algorithms. https://cp-algorithms.com/algebra/discrete-root.html. url consultata il 09-05-2019.

D.P. DiVincenzo. Topics in quantum computers. https://arxiv.org/abs/cond-mat/9612126, 1996. url consultata il 02-06-2019.

finite fields - Wikipedia. Finite field - wikipedia. https://en.wikipedia.org/wiki/Finite_field, ultima modifica 23-05-2019. url consultata il 06-06-2019.

First quantum bit. Breakthrough in bid to create first quantum computer. https://newsroom.unsw.edu.au/news/technology/breakthrough-bid-create-first-quantum-computer, 2012. url consultata il 02-06-2019.

Group - Wikipedia. Group (mathematics) - wikipedia. https://en.wikipedia.org/wiki/Group_(mathematics), 2002. url consultata il 10-06-2019.

Larry Hardesty. 3q: Scott aaronson on google's new quantum-computing paper. http://news.mit.edu/2015/3q-scott-aaronson-google-quantum-computing-paper-1211, 2015. url consultata il 02-06-2019.

Matthew Hayward. Quantum computing and shor's algorithm. https://quantum-algorithms.herokuapp.com/299/paper/index.html, originale 1999, modificate nel 2015. url consultata il 28-05-2019.

W. Cary Huffman and Vera Pless. *Fundamental of Error Correcting Codes*. Cambridge University press, 2003.

IBM cloud computing. Ibm makes quantum computing available on ibm cloud to accelerate innovation. https://www-03.ibm.com/press/us/en/pressrelease/49661.wss, 2016. url consultata il 02-06-2019.

Ellen Jochemsz. Goppa codes and the mceliece cryptosystem. https://klevas.mif.vu.lt/~skersys/vsd/crypto_on_codes/goppamceliece.pdf, 2002. url consultata il 20-05-2019.

Khan Academy. Journey into cryptography-computer science-computing-khan academy. https://www.khanacademy.org/computing/computer-science/cryptography. url consultata il 17-04-2019.

Frederic Lardinois. Ibm unveils its first commercial quantum computer. https://techcrunch.com/2019/01/08/ibm-unveils-its-first-commercial-quantum-computer/, 2019. url consultata il 03-06-2019.

Enrique Martin-Lopez, Anthony Laing, Thomas Lawson, Roberto Alvarez, Xiao-Qi Zhou, and Jeremy L. O'Brien. Experimental realization of shor's quantum factoring algorithm using qubit recycling. https://arxiv.org/abs/1111.4147, 2011. url consultata il 02-06-2019.

Alasdair McAndrew. *Introduction to Cryptography with open-source software*. CRC press, 2012. parte della collana 'Discrete Mathematics and Its Applications'.

Hartmut Neven. Launching the quantum artificial intelligence lab. https://ai.googleblog.com/2013/05/launching-quantum-artificial.html, 2013. url consultata il 02-06-2019.

Numeri RSA - Wikipedia. Numeri rsa- wikipedia. https://it.wikipedia.org/wiki/Numeri_RSA. url consultata il 08-05-2019.

Okalhoma State University. Congruence and congruence classes. http://lie.math.okstate.edu/ binegar/3613/3613-l11.pdf, 2005. url consultata il 02-05-2019.

Online calculator. Online calculator: Linear diophantine equations. https://planetcalc.com/3303/. url consultata il 28-06-2019.

Order of finite fields. Order of finite fields is p^n-mathematics stack exchange. https://math.stackexchange.com/questions/72856/order-of-finite-fields-is-pn. url consultata il 18-05-2019.

p-group - Wikipedia. p-group - wikipedia. https://en.wikipedia.org/wiki/P-group, 2001. url consultata il 09-06-2019.

Robert Perkins. Robert j. mceliece, 1942–2019. https://www.caltech.edu/about/news/robert-j-mceliece-19422019, 2019. url consultata il 06-06-2019.

Quantum Computing- Wikipedia. Quantum computing - wikipedia. https://en.wikipedia.org/wiki/Quantum_computing#Timeline, ultima modifica: 2019. url consultata il 03-06-2019.

Quantum Decoherence - Wikipedia. Quantum decoherence - wikipedia. https://en.wikipedia.org/wiki/Quantum_decoherence, ulitma modifica il 23-04-2019. url consultata il 02-06-2019.

Quantum register - Wikipedia. Quantum register - wikipedia. https://en.wikipedia.org/wiki/Quantum_register, ultima modifica: 2018. url consultata il 03-06-2019.

Christopher Roering. Coding theory-based cryptopraphy: Mceliece cryptosystems in sage. http://digitalcommons.csbsju.edu/honors_theses/17, 2013. url consultata il 08-06-2019.

A. J. Di Scala. Corso di crittografia, lezione 1. Appunti delle lezioni, 2019.

Mark Sevalnev. Error-correcting codes introduction, hamming distance. http://mathworld.wolfram.com/Error-CorrectingCode.html, 2008. url consultata il 06-06-2019.

IBM Research Editorial Staff. Quantum computing: Breaking through the 49 qubit simulation barrier. https://www.ibm.com/blogs/research/2017/10/quantum-computing-barrier/, 2017. url consultata il 02-06-2019.

A. P. Stakhov. Fibonacci matrices, a generalization of the "cassini formula", and a new coding theory. *Chaos, Solitons and Fractals 30*, pages 56–66, 2006.

teorema Bolzano - Wikipedia. Teorema di bolzano - wikipedia. https://it.wikipedia.org/wiki/Teorema_di_Bolzano, 2006. url consultata il 28-06-2019.

Lieven M.K. Vandersypen, Matthias Steffen, Gregory Breyta, Costantino S. Yannoni, Mark H. Sherwood, and Isaac L. Chuang. Experimental realization of shor's quantum factoring algorithm using nuclear magnetic resonance. https://arxiv.org/abs/quant-ph/0112176, 2001. url consultata il 02-06-2019.

Eric W. Weisstein. One-way function. from mathworld–a wolfram web resource. http://mathworld.wolfram.com/One-WayFunction.html. url consultata il 09-06-2019.

Wikipedia. Equazione diofantea lineare. https://it.wikipedia.org/wiki/Equazione_diofantea_lineare. url consultata il 18-04-2019.

Nanyang Xu, Jing Zhu, Dawei Lu, Xianyi Zhou, Xinhua Peng, and Jiangfeng Du. Quantum factorization of 143 on a dipolar-coupling nmr system. https://arxiv.org/abs/1111.3726, 2011. url consultata il 02-06-2019.