Università degli studi di Modena e Reggio Emilia

Dipartimento di Ingegneria Enzo Ferrari

Algoritmi di Crittografia

Anno Accademico 2023/24

Indice

1 Crittografia **1**

1.1 Introduzione 1

1.2 Cifratura, decifrazioni e attacchi 1

1.3 Livelli di Segretezza 3

1.4 La chiave Pubblica 5

2 Crittografia Simmetrica **7**

2.1 Cifrari Classici: Cesare e Vigenère 8

2.2 One-time pad - Cifrario di Vernam 13

2.3 Obiettivi generali di sicurezza 14

2.4 Cifrari a Blocchi 15

2.4.1 Feistel Network 16

2.5 Mode of Operation 19

2.5.1 Electronic Codebooks Mode - ECB 19

2.5.2 Cipher Block Chaining - CBC 19

3 Aritmetica Modulare **21**

3.1 Algoritmi Probabilistici 34

3.1.1 Algoritmi di Monte Carlo 34

3.1.2 Algoritmi di Las Vegas 35

3.2 Teoria e Algoritmi per il Test di Primalità 36

3.3 Test di Primalità di Solovay - Strassen 38

3.4 Algoritmo di Fattorizzazione ρ di Pollard 39

4 Crittografia Asimmetrica e Scambio di Chiavi **41**

4.1 Diffie-Hellman 41

4.2 ElGamal 45

4.3 Sistema Crittografico RSA 48

4.4 Sistema Crittografico Asimmetrico di Rabin 56

5 Firma Digitale **61**

5.1 Firma Digitale con RSA 61

5.2 Firma Digitale con ElGamal 64

5.3 Firma Digitale con Digital Signature Algorithm 67

5.4 Autenticità delle chiavi pubbliche 68

5.4.1 Approccio TLS 68

5.4.2 GnuPG 69

6 Curve Elittiche **70**

6.1 Campi Finiti 71

6.2 Curva Elittica 72

6.2.1 Curve Elittiche Smooth 73

6.2.2 Curve Elittiche non Smooth 74

6.3 Operazioni Geometriche su una Curva 74

6.4 Curve in $\mathbb{Z}_p$ 82

7 Algoritmi di Crittografia su elittica **85**

7.1 Elliptic Curve Diffie-Hellman 87

7.2 Elliptic Curve - Digital Signature Algorithm 89

Capitolo 1

Crittografia

1.1 Introduzione

La *Crittografia* (etimologicamente "scrittura segreta") ha lo scopo di mascherare messaggi in modo da poter permettere lo scambio di informazioni tra due interlocutori, che per comodità chiameremo **Alice** e **Bob**, senza che un possibile "origliatore", **Eve**, possa comprendere il messaggio scambiato.
Se da una parte sono presenti attori come Alice e Bob che cercano di mandarsi messaggi segreti tramite *metodi di cifratura*, avremo per definizione altri attori, identificati in Eve, che attraverso *metodi di crittoanalisi* cercheranno di riportare alla luce le informazioni contenute in quelle conversazioni.

1.2 Cifratura, decifrazioni e attacchi

La **Crittologia** è la scienza che raggruppa lo studio della *crittografia* con i suoi metodi di cifratura e lo studio di come poter rompere questi schemi crittografici ovvero la *crittoanalisi*. Il problema che si vuole risolvere attraverso la crittografia è il seguente: un *mittente*, Alice, vuole comunicare con un *destinatario*, Bob, utilizzando un canale di trasmissione *insicuro*, ovvero che intercettando un messaggio è possibile leggerlo, comprenderlo e modificarlo.

Definiamo **MSG** come l'insieme a cui appartengono i messaggi, mentre **CRT** come l'insieme a cui appartengono i crittogrammi andiamo a definire:

- **Cifratura del Messaggio**: operazione per la quale si trasforma un generico messaggio in chiaro $\boldsymbol{m}$ in un crittogramma $\boldsymbol{c}$ applicando una funzione $C : MSG \rightarrow CRT$
- **Decifrazione del Messaggio**: operazione che permette di ricavare il messaggio in chiaro $\boldsymbol{m}$ dal crittogramma $\boldsymbol{c}$ applicando una funzione $D : CRT \rightarrow MSG$

Matematicamente $D(C(m)) = m$ infatti le funzioni $\boldsymbol{D}$ e $\boldsymbol{C}$ sono una l'inverso dell'altra. Sempre per definizione la funzione $\boldsymbol{C}$ deve essere *iniettiva*, ovvero, a messaggi diversi devono corrispondere crittogrammi diverssi, altrimenti Bob non potrebbe ricostruire univocamente un messaggio da ogni crittogramma.
Analizzando ora il punto di vista di Eve, ovvero di un crittoanalista che si inserisce nella comunicazione. La tipologia di azioni che va a svolgere possono essere di due tipologie: *passivo* ovvero se Eve si limita ad ascoltare la comunicazione, o *attivo* se agisce sul canale disturbando la comunicazione o alterandola.
L'attacco a un sistema crittografico dipende dalle informazioni in possesso da un crittoanalista:

- ***Cipher Text Only Attacker (COA)***: il crittoanalista è riuscito a leggere dal canale insicuro una serie di crittogrammi $c_1, c_2, ..., c_n$.
- ***Known Plain-Text Attack (KPA)***: il crittoanalista è venuto a conoscenza di usa serie di coppie $(m_1, c_1), (m_2, c_2), ..., (m_n, c_n)$ contenenti i messaggi in chiaro e il crittogramma corrispondente.
- ***Chosen Plain-Text Attack (CPA)***: il crittoanalista ha ottenuto una serie di coppie $(m_1, c_1), (m_2, c_2), ..., (m_n, c_n)$ di cui ha scelto il testo in chiaro. Effettuando delle **query** di **encryption**. Normalmente questa casistica fa riferimento alla *crittografia asimmetrica*, in cui il crittanalista conosce la chiave di cifratura.
- ***Chosen Cipher-Text Attack (CCA)***: invece che un **encryption query** il crittanalista ha la possibilità di eseguire una **decryption query**.

- ***Man in the Middle***: il crittoanalista riesce ad porsi all'interno del canale di comunicazione interrompendo le comunicazioni tra Alice e Bob e alterando i messaggi, convincendo entrambi che tali messaggi provengano legittimamente dall'altro.

Dopo aver definito le diverse tipolgie di attacco che può utilizzare un crittoanalista, andiamo a definire anche, i **requisiti tipici di sicurezza** in una comunicazione:

- **Confidenzialità**: ovvero che Eve non sia in grado di comprendere il contenuto del messaggio che viene trasmesso su un canale insicuro.
- **Autenticazione**: ovvero la sicurezza per la quale Bob è sicuro che il messaggio sia effettivamente inviato da Alice.
- **Integrità**: ovvero la sicurezza per la quale Bob è sicuro che Eve non abbia manomesso il messaggio.
- **Non Ripudio**: ovvero la caratteristica per la quale Alice non può, negare, nemmeno in un secondo momento, di aver inviato il messaggio a Bob.

1.3 Livelli di Segretezza

Il grosso passaggio nella crittografia avvenne con l'avvento dei calcolatori elettronici, fino a quel momento la sicurezza della crittografia risiedeva non in una chiave di cifratura, ma nel mantenere segrete le funzioni $\boldsymbol{C}$ e $\boldsymbol{D}$, in quanto la cifratura e la decifratura venivano eseguite a mano. Erano quindi stati stabilit delle *norme* per caratterizzare la "robustezza" di un cifrario, ovvero: le funzioni $\boldsymbol{C}$ e $\boldsymbol{D}$ dovevano essere *facili* da calcolare, era *impossibile* ricavare $\boldsymbol{D}$ se $\boldsymbol{C}$ non era nota e il crittogramma $c = C(m)$ doveva apparire innocente.
Ovviamente con l'avvento dei calcolatori eletronnici, bisogna rimodulare queste caratteristiche, infatti il crittogramma viene inviato come sequenza di 0 e 1 ed è quindi "sempre innocente". L'altra questione è che tenere segreto l'intera funzione di cifrazione inizia ad essere impensabile, nascono quindi cifrari che si basano sulla segretezza di una chiave k che permette, mandata in input alla funzione di cifrazione di ottenere il

crittogramma. Utilizziamo quindi la notazione: $C(m, k)$ per la cifratura e $D(c, k)$ per la decifrazione.
Naturalmente occorre presare molta attenzione alla scelta della chiave, ma anche in questo caso, **Kerckhoffs** enuncia un **principio fondamentale** attraverso il quale trovare una chiave adatta:

Se le chiavi sono: scelte bene, di lunghezza adeguata, tenute segrete e gestite unicamente da sistemi fidati allora non ha importanza mantenere segreti anche gli algoritmi di crittografia.

Avere gli algoritmi di crittografia pubblici permette di individuare in tempi ridotti eventuali debolezze.
Uno dei grossi "problemi" della crittografia classica è che il testo viene si cifrato, ma viene mantenuta sia la struttura del messaggio ma anche le ricorrenze delle varia lettere. É quindi possibile attraverso la **crittoanalisi statistica** risalire al testo in chiaro. Nella crittografia simmetrica vengono introdotte dei *pattern* per i quali si possono superare i problemi della crittoanalisi statistica, il **Modello di Shannon** descrive queste caratteristiche come:

- **Diffusione**: ogni bit del testo cifrato deve essere influenzato da molti bit del testo in chiaro, è ottenibile applicando **permutazioni** sui dati in ingresso più una *funzione*: l'effetto è che più bit in posizioni differenti nel testo in chiaro influenzino il valore di un bit del testo cifrato.

- **Confusione**: bisogna che non si possa risalire alla chiave del testo cifrato. Questo si ottiene tramite un algoritmo di **Sostituzione**.

Vantaggi	**Svantaggi**
Efficenza e Velocità di Elaborazione: basandosi su algoritmi meno complessi permette di utilizzare acceleratori hardware per la computazione	**Scambio Sicuro delle Chiavi**: Richiede un canale sicuro di scambio delle chiavi. Se un agente malevolo ottiene le chiavi può decifrare i dati.
Robustezza: se implementato correnttamente e con una lunghezza della chiave appropriata è computazionalmente sicuro.	**Scalabilità dei sistemi**: gestire un alto numero di chiavi simmetriche (una per ogni iterlocutore con cui si deve comunicare) può diventare complicato e oneroso sulla memoria del sistema.
	Nessuna Autenticazione: La crittografia non affronta direttamente il problema dell'**autenticazione** delle parti coinvolte.

1.4 La chiave Pubblica

Nel 1976 venne introdotta la *crittografia a chiave pubblica*, con l'obiettivo di permettere a tutti di inviare messaggi cifrati, ma di abilitare solo il destinatario a decifrarli. Nel concetto di **crittografia a chiave pubblica** le operazioni di cifratura e decifratura utilizzano due chiavi diverse *k[pub]* per cifrare e *k[prv]* per decifrare. La prima è pubblica e quindi non è necessaria in "locale", andando così a liberare della memoria, mentre la seconda è privata quindi nota solo al destinatario.
Bisogna andare a modifica anche la funzione di cifratura $\boldsymbol{C}$ in quando bisognerà che possegga una proprietà detta *one-way trap-door*. Ovvero che il calcolo di $c = C(m, k[pub])$ per la cifratura sia computazionalmente facile, mentre il calcolo inverso $m = D(m, k[prv])$ per la decifrazione sia computazionalmente oneroso (*one-way*), a meno che non si conosca un parametro che permetta una risoluzione meno onerosa, in questo caso è il ruolo della *k[prv]*. La difficoltà è però condizionata a congetture matematiche universalmente accettate ma non dimostrate esplicitamente, per cui tali funzioni saranno utilizzabili solo fino al momento in cui se ne dimostri una semplice invertibilità. Le due funzioni che vengono utilizzate principalmente sono: l'**esponenziale**

modulare (funzione *one-way*) e la sua inversa ovvero il **logaritmo discreto**, oppure, la **moltiplicazione modulare** e la sua inversa, ovvero la **fattorizzazione**.
Riuscire a rompere un sistema crittografico che utilizzi o l'esponenziale modulare o la moltiplicazione modulare implica che a livello matematico si è riusciti a trovare un algoritmo che permetta la computazione di logaritmo discreto e fattorizzazione in maniera computazionalmente facile.
L'introduzione di cifrari asimmetrici ha avuto alcune importanti conseguenze, in quanto un cifrario simmetrico con n utenti sono necessarie $n(n-1)/2$ chiavi segrete, una per ogni coppia di utenti. Mentre in un cifrario asimmetrico sono sufficienti n coppie di chihavi publlica/privata, e scompare il loro scambio segreto perché le chiavi di cifratura sono pubbliche, e ciascuna chiave di decifrazione è in possesso di un solo utente che la manitene segreta.
A contrastare questo grande vantaggio che portano i protocolli asimmetrici, di contro sono molto più lenti a livello computazionale rispetto ad un protocollo simmetrico che rimane lo standar per comunicazioni di massa. Viene quindi adottata una struttura di crittografia *ibrida* in cui viene utilizzata la crittografia asimmetrica per lo scambio di **chiavi segrete** e poi di crittografia simmetrica per l'intera durata della comunicazione.

Capitolo 2

Crittografia Simmetrica

Nella struttura di un cifrario simmetrico si possono individuare due distinte componenti. Il primo è l'**algoritmo**, o semplicemente l'insieme di operazioni che applicano una determinata **permutazione** ad una porzione di *plain text*. Il termine "porzione" è definito ad una caratteristica del cifrario. Infatti può essere una *singola lettera*, un *gruppo di lettere* o (come nei cifrari moderni) un *gruppo di bit* detto comunemente **blocco**. Una **permutazione** di un insieme di n oggetti è uno dei possibili modi in cui gli n oggetti si possono disporre linearmente; cioè è uno dei possibili ordinamenti degli oggetti. In termini matematici, una permutazione di un insieme I è una funzione invertibile in I stesso. Si noti che l'ovvio requisito di invertibilità delle cifratura, che si precisa fondamentalmente con l'invertibilità delle permutazioni, in termini pratici implica che il plaintext e ciphertext abbiamo sempre la **stessa lunghezza**. La seconda componente di un algoritmo è il **mode of operation** ovvero come le "porzioni" di plaintext vengono opportunamente concatenate una con l'altra.
Quelli appena descritti sono però definiti **cifrari a blocchi**. Esistono in letteratura cifrari che considerano il messaggio come *flusso continuo* di bit e che utilizzano la chiave per generare un corrispondente flusso di bit, nota come **round key** che verrà posta in XOR con i bit del messaggio. Tali cifrari sono noti come ***stram cipher***, vengono normalmente utilizzati nelle comunicazione telefonoiche, ad esempio.

2.1 Cifrari Classici: Cesare e Vigenère

Considerando i messaggi scritti in un generico alfabeto Σ e consideriamo un **ordinamento circolare** dei caratteri, ovvero che il primo è il successore dell'ultimo. Nel **Cifrario di Cesare** la cifratura avviene:

1. Viene applicata una **permutazione** ad ogni carattere del messaggio che consiste in uno **shift** (slittamento) di k posizioni in avanti.

2. In questo modo ogni carattere x del plaintext viene sostituito con il carattere $x + k$ rispetto all'ordinamento circolare di Σ.

3. In questo caso il **mode of operation** è applicare la stessa permutazione ad ogni carattere.

In questo caso la **chiave privata** non è altro che k, ovvero il valore dello slittamento. In questo tipo di cifrario la **decifrazione** non è altro che uno **shift** di k posizione di ogni carattere del ciphertext all'indietro. Considerando un carattere del ciphertext x' avremo che $x = x' - k$ per ogni carattere del ciphertext. Possiamo anche dire che se la chiave di cifratura è k allora la chiave di decifrazione è $-k$ ovvero il suo opposto. In questo semplice cifrario il **numero di chiavi**, che coincide con il numero di possibili slittamenti, è limitato dalla cardinalità dell'alfabeto.
Prendiamo ora in considerazione ora un esempio di crittoanalisi per il cifrario di Cesare. Consideriamo l'alfabeto $\Sigma = \{a, b, .., z\} \cup \{.\} \cup \{,\} \cup \{'\}$ e considerando come **cipher** = ***kdcxiitzz,ev,yhtjeved,bv,yhth,ew,vxithx*** potremmo utilizzare una procedura di crittoanalisi di **brute force** (forza bruta) ovvero tentare per tutti i possibili valori di k quello cha ha per risultato un testo sensata. Considerando il codice 2.1 possiamo andare ad iterare su tutti i possibili valori di k e farci stampare a video il risultato di ogni iterazione in questo modo è possibile andare a leggere tutte le possibili permutazioni di x e andare a decifrare il contenuto del messaggio.
Possiamo notare che per $k = 19$ si può leggere il messaggio *unmessaggiocifratoconilcifrariodicesare* che interpretandolo correttamente diviene *un messaggio cifrato con il cifrario di cesare*, il fatto di rimuovere gli spazi serve per evitare di dare al crittoanalista un'informazione, ovvero la lunghezza di ogni parola.

```
1   alphabet = "abcdefghijklmnopqrstuvwxyz.,'"
2
3   def decrypt(cyphertext, key):
4       n = len(alphabet)
5       cyphertext = ''.join([alphabet[(alphabet.find(c) + key) %
            n] for c in plaintext])
6
7   unknowntext = "kdcxiitzz,ev,yhtjeved,bv,yhth,ew,vxithx"
8   for k in range(len(alphabet)):
9       print(f"{k}\t{decrypt(unknowntext, -k)}")
10
11  # output
12  19  unmessaggiocifratoconilcifrariodicesare
```

Il **cifrario di Cesare** è forse il più semplice caso di **cifrario mono-alfabetico** infatti utilizza una sostituzione fissa per ogni carattere, in altri termini, fissato l'alfabeto di riferimento Σ, un *cifrario mono-alfabetico* è definito da una **singola permutazione** ripetuta per tutti i caratteri. Un'altra caratteristica del cifrario di cesare è che la **permutazione** è una semplice **rotazione** e per riuscire a rompere questo cifrario è necessario la conoscenza di un singolo numero intero, compreso fra 0 e la cardinalità dell'alfabeto. La chiave è l'unica **informazione segreta** di un cifrario mono-alfabetico e quindi il testo cifrato deve dipendere sempre dalla permutazione. A chiavi diverse bisogna che ci siano diversi ciphertext, in caso contrario per un attaccante ci sarebbero meno chiavi distinte da provare e questo potrebbe agevolare gli attacchi. La permutazione deve apparire il più possibile casuale. Questo garantisce che la conoscenza di come viene mappata una lettera (o un gruppo di bit) non fornisce alcuna conoscenza sulla mappatura delle altre.

Noto il fatto che per calcolare le possibili permutazioni di una certa lunghezza l'operazione è il fattoriale avremo un numero di permutazione tali a: $n_p = card(\Sigma)!$

```
1  alphabet = "abcdefghijklmnopqrstuvwxyz .,'"
2  print(factorial(len(alphabet)))
3
4  # output
5  265252859812191058636308480000000
```

Crittoanalisi: per un generico cifrario mono-alfabetico il numero di possibili chiavi è dunque pari al numero di permutazioni dell'insieme dei caratteri dell'alfabeto, in questo caso un attacco *brute force* è impensabile, nel caso di cifrari mono-alfabetici la crittoanalisi più efficace è quella bata sull'**analisi delle frequenze**. L'attacco si basato sull'ipotesi che la frequenza con cui compaiono i singoli caratteri nel plaintext sia conforme alla più generale **frequenza nella lingua** con cui il testo è scritto, questo mette come vincolo il fatto di conoscere la lingua del testo cifrato che si vuole decodificare e l'ipotesi sarà tanto più verificata quanto più il plaintext è lungo. Un'altro vincolo è quello di disporre di una **stima delle frenquenze dei caratteri** in quella lingua.

Vigenère: un cifrario poli-alfabetico, ovvero che uno stesso carattere non è sempre soggetto alla stessa trasformazione. Le chiavi sono **sequenze di caratteri dell'alfabeto** Σ, ciascuno dei quali deve essere interpretato come il numero corrispondente alla sua **posizione nell'alfabeto** (partendo da 0). Ad esempio se il palintext fosse **algorithms** e la chiave fosse **crypto**, il ciphertext verrebbe determinato in questo modo:

key :	c	r	y	p	t	o	c	r	y	p
key_i :	2	17	24	15	19	14	2	17	24	15
plaintext :	a	l	g	o	r	i	t	h	m	s
$plaintext_i$:	0	11	6	14	17	8	19	7	12	18
ciphertext :	c	c	e	d	k	w	v	y	k	h
$ciphertext_i$:	2	2	4	3	10	22	21	24	10	7

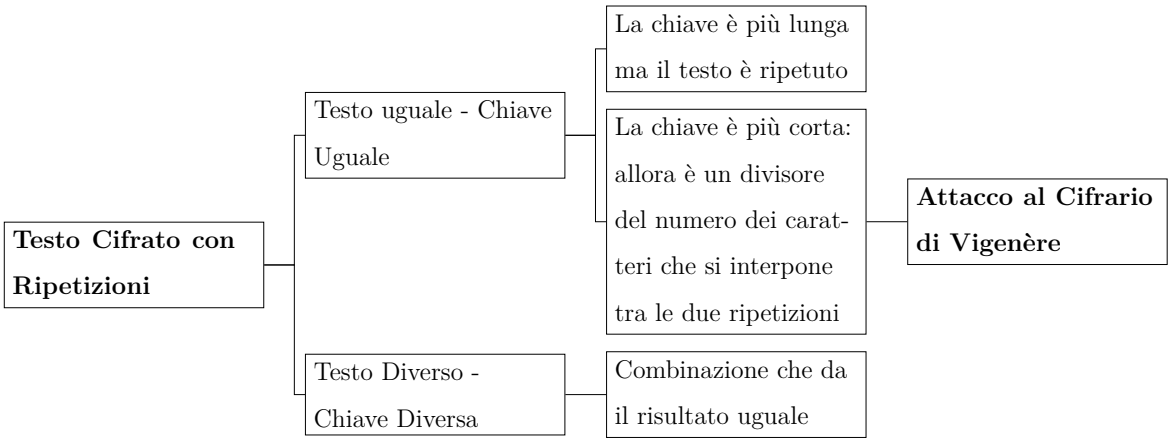
Avremo quindi una funzione di encryption del tipo:

```
1   alphabet = "abcdefghijklmnopqrstuvwxyz"
2   plain, key = "algorithms", "crypto"
3
4   a = {alphabet[i]: i for i in range(len(alphabet))}
5   b = {i: alphabet[i] for i in range(len(alphabet))}
6
7   def encryption(plain, key):
8       cipher = ""
9       for i in range(len(plain)):
10          x = a.get(plain[i])
11          k = a.get(key[i % len(key)])
12
13          cipher += b.get((x + k) % len(alphabet))
14      return cipher
15
16      print(f"[ PLAIN ]: {plain}, [ CIPHER ]: {encryption(plain,
            key)}")
17
18  # output
19  [ PLAIN ]: algorithms, [ CIPHER ]: ccedkwvykh
```

Per la **decifrazione** si procede allo stesso modo utilizzando però, sempre con la stessa chiave, il valore di slittamento all'indietro.

Attacchi al cifrario di Vigenère: senza l'ausilio di un computer questo tipo di cifrario era praticamente inattaccabile, soprattuto con una chiave casuale, tutt'oggi è difficilmente attaccabile in caso di chaive casuale con lunghezza confrantabile con il testo e non riutilizzata più volte. In quest'ultimo caso, il cifrario "degenera" infatti in una sorta di **one-time pad testuale** noto anche come ***cifrario di Vernam***. Nel caso in cui invere la chiave è relativamente corta rispetto al testo è possibile effettuare un attacco la cui prima parte è quella di indentificare la **lunghezza della chiave**.

Consideriamo, come esempio, un testo cifrato $\mathbf{C} = \boldsymbol{fxbxzktwrdvfcsfxbxqwplbn}$ e notiamo che la sequenza di 4 caratteri **fxbx** si ripete ad una distanza di 14 posizioni. Potremo quindi iniziare a fare dei check sulle possibilità di crittoanalisi per questo ciphertext.



Una volta che ipotizziamo di essere nella casistica corretta per effettuare un attacco di crittografia, *Eve* può provare ad indovinare la lunghezza della chiave, che può essere un divisore di 14: 2, 7, 14. Considerando 2 come chiave troppo corta e 14 come chiave troppo lunga, proviamo a portare avanti l'attacco considerando la chiave lungha 7 caratteri. A questo punto *Eve* può tentare almeno tre strade:

1. **Attacco brute force**: computazionalmente non troppo oneroso, infatti le chiavi possibili sono soltanto $26^7 = 8031810176$.

2. **Analisi delle frequenze**: se il testo è abbastanza lungo si può provare a stimare le frequenze considerando che due identici caratteri del testo cifrato corrispondono con certezza allo stesso carattere del plaintext se e solo se la loro posizione differisce di un multiplo di 7.

3. **Attacco a Dizionario**: ipotizzando che la chiave non sia casuale, *Eve* può provare tutte le parole di 7 lettere di un dizionario (della lingua da cui proviene il plaintext).

2.2 One-time pad - Cifrario di Vernam

Il **One-time pad** è "teoricamente" il cifrario che ha una **sicurezza incondizionata** ovvero il testo cifrato generato non contiene abbastanza informazioni per determinare il testo in chiaro, indipendentemente da quanto testo cifrato sia disponibile, è computazionalmente molto oneroso per essere utilizzato. Il funzionamento consiste nel prendere un messaggio (plaintext) $\boldsymbol{P}$, ovvero una *string di bit*, e una chiave che è una sequenza di bit della *stessa lunghezza del testo in chiaro*. Avremo quindi che:

$$c_i = p_i \oplus k_i, \qquad i = 0, 1, \ldots, |P| - 1$$

Ovvero che il testo cifrato, ciphertext, sarà lo *xor* bit a bit tra il plaintext e la chiave. Affinché la chiave sia adatta bisogna che rispetti una serie di condizioni:

- Ogni bit della chiave deve avere valore 0 oppure 1 con **uguale probabilità** e in modo **indipendente** dal valore dei bit precedenti, ovvero:

$$prob[k_i = 0] = prob[k_i = 1] = \tfrac{1}{2}$$
$$prob[k_i = v | k_{i-1} = w] = prob[k_i = v],\ v, w \in \{0, 1\}$$

- La chiave deve essere **utilizzata una sola volta**, è quindi necessario generare una nuova chiave ogni volta che si vuole inviare un messaggio.

É quindi possibile osservare che per i bit del messaggio cifrato valgono le stesso proprietà, indipendentemente da come sono scelti i bit del messaggio ovvero del plaintext. La **sicurezza del one-time pad** deriva dal fatto che se la chiave è stata scelta come indicato sopra ed è di lunghezza n (quindi n sarà anche la lunghezza di P) ogni messaggio ha una possibiltà pari 2^{-n} di essere decriptato. Nel caso in cui due messaggi in chiaro P_1 e P_2 vengono cifrati con la stessa chiave K allora è possibilie ottenere della conoscenza sulla comunicazione.

$$\begin{aligned} C_1 \oplus C_2 &= (P_1 \oplus K) \oplus (P_2 \oplus K) \\ &= (P_1 \oplus P_2) \oplus (K \oplus K) \\ &= (P_1 \oplus P_2) \oplus O \\ &= P_1 \oplus P_2 \end{aligned}$$

In quyesto modo l'attaccante viene a conoscenza dello **xor dei due messaggi** e dunque, conoscendone uno dei due (o parte di uno dei due) verrebbe a conoscenza anche (parte de)l'altro.

2.3 Obiettivi generali di sicurezza

Esistono più obiettivi di un attacco da parte di *Eve*, di cui "capire il messaggio" non è neppure il più estremo, potrebbe essere che voglia **determinare la chiave** che permetterebbe di mettere in chiaro uno o più messaggi o un altro consiste nel far passare per autentico un messaggio che invece *Eve* ha **alterato** o prodoto nella sua interezza. É quindi necessario distinguere i due principali obiettivi:

- ***Indistinguibilità***: è la proprietà di uno schema di cifratura i cui testi cifrati per un attaccante sono essenzialmente *indistinguibili* da stringhe di bit generate casualmente. Facciamo un esempio: *Alice* e *Eve* sono gli attori, *Eve* produce **due messaggi in chiaro** e li sottopone ad *Alice*, che deciderà, a caso e con uguale probabilità, quale dei due cifrare, e presenterà quello cifrato ad *Eve*, il cifrario è indistinguibile se *Eve* non riesce ad indovinare quale messaggio sia stato cifrato con probabilità significativamente maggiore di $\frac{1}{2}$.

- ***Non malleabilità***: ovvero se, dato un ciphertext C_1 corrispondente ad un plaintext P_1, risulta impossibile per *Eve* creare un secondo ciphertext C_2 corrispondente ad un plaintext P_2 che abbia una qualche relazione con P_1. La non malleabilità è una proprietà che ha a che fare con l'**integrità** dei messaggi.

Nel caso del *one-time pad* si può affermare che è soa indistinguibile che non malleabile a meno che una stessa chiave venga utilizzata per due o più plaintext diversi.

2.4 Cifrari a Blocchi

I cifrari a blocchi moderni vengono definiti a blocchi in quanto lavorano su blocchi di lunghezza fissa, indichiamo con B il numero di bit di un blocco, che normalmente è compreso nell'intervallo tra i 64 bit e i 256 bit. Nel caso in cui il plaintext da cifrare abbia una lunghezza superiore rispetto a B allora il plaintext verrà diviso in blocchi con lunghezza pari a B, ma nel caso in cui l'ultimo blocco o l'ìntero plaintext abbia lunghezza inferiore a B verranno aggiunti **bit di padding** (riempimento). Nel caso in cui il plaintext venga diviso in sottoblocchi che verranno cifrati separatamente sarà necessario uno schema di concatenazione di vari blocchi, questo è identificato come **mode of operation**.
Sia padding che mode of operation sono tutt'altro che banali utilizzando schemi obsoleti o "sbagliati" nella circostanza di criptazione dare vantaggi ai crittoanalisti per la decifrazione malevola del crittogramma.
É quindi necessario scegliere correttamente non solo il mode of operation e lo standard di padding che si vuole adottare ma anche considerare la lunghezza di B in quanto bisogna trovare un valore per cui B non sia troppo lungo con lo scopo di incrementare l'efficienza (**hardware dedicato**), ma non deve essere neanche troppo corto per evitare problemi legati alla sicurezza ma anche ridurre l'overhead legato al trattamento di molti messaggi corti. Nel caso in cui un blocco sia lungo B piccolo, un attaccante potrebbe precompilare una tabella 2^B posizioni in cui alla generica posizione i è memorizzato il plaintext che ha i come ciphertext

Codebook Attack: nel caso di B piccolo corrisponderebbe ad un semplice **table lookup**, altre tipologie di attacco che invece potrebbero utilizzare ad esempio, il tempo di esecuzioni di operazioni su chiave privata vengono chiamati ***Side Channel Attack***.

Possiamo andare a descrivere in maniera astratta uno ***Schema di Block Cipher***: andiamo a definire che una chiave k necessaria sia per la cifrazione che decifrazione ha normalmente una lunghezza compresa tra **128 e 256 bit**, e abbiamo assimilato che la cifratura non è altro che una **permutazione** poiché plaintext e ciphertext sono formati dallo stesso numero di bit. É anche noto che ad ogni plaintext di B bit deve

corrispondere **un unico** ciphertext di B bit, e viceversa, altrimenti il processo non sarebbe invertibile. Potremmo visualizzare la cifratura come un processo di **due fasi**:

- data la chiave k, si seleziona una specifica ***Lookup Table*** T_k, di 2^B posizioni, che memorizza una specifica permutazione delle sequenze di B bit.
- la cifratura corrisponde ad un accesso a T_k usando il plaintext come chiave su quella tabella e riuscendo a recuperare il ciphertext.

É immediato osservare che la lunghezza della chiave k determini il **numero delle lookup table** e dunque di permutazioni, che possono essere utilizzate in un cifrario a blocchi con chiavi di quella lunghezza. Se dunque la chiave è lungha m bit, allora il numero di permutazioni usabili è 2^m. Questo valore è apparentemente grande infatti considerando che le permutazioni totali di B bit, che cresce in maniera molto più repentina ovvero: $2^B!$. Sono presenti però delle permutazioni di B bit che però vengono scartate, ovvero quelle che possono essere espresse **mediante trasformazioni lineari affini invertibili** sul campo Z_2.

$$xP + b = y(x)$$

in cui P è una matrice non singolare di ordine B su Z_2 mentre x (il plaintext), b e $y(x)$ (il ciphertext) sono vettori riga di B element, sempre con elementi in Z_2.

Un cifrario a blocchi ha come scopo, alla fine, quello di **forzare la non-linearità** della permutazione realizzato tramite i concetti di:

- ***Diffusione***: ogni minima modifica localizzata in un punto del plaintext, si rifletta su tutto il ciphertext.
- ***Confusione***: è la caratteristica che **distrugge ogni regolarità**, ogni eventuale pattern nel plaintext.

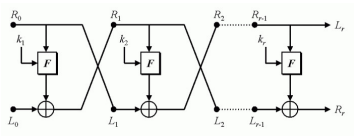
2.4.1 Feistel Network

La rete di Feistel non è un vero e proprio cifrario a blocchi, ma si tratta di uno schema per realizzare cifrari a blocchi, fra i quali è presente **Digital Encryption Standard**

(***DES***) che rimase per molti anni lo standard per la cifratura simmetrica, ora deprecato e sostituito da **Advanced Encryption Standard** (***AES***). La rete di Feistel unisce i concetti di cifrario prodotto a quelli di Shannon, ovvero *diffusione* e *confusione*, riuscendo ad approssimare il funzionamento di un cifrario a blocchi ideale attraverso:

- l'alterazione di tecniche di **sostituzione** a quelle di **permutazione** per la confusione dei bit.
- **chiave** a k bit per un blocco a n bit ($2^k << 2^n$).

La rete di Feistel è composta da un certo numero di **stadi con identica struttura** noti anche come **round**. Ad ogni stadio viene utilizzata una specifica **round key** k_i, una diversa per ogni stadio sono tutte derivabili dalla **chiave generale**, questo avviene tramite un algoritmo (ovviamente reversibile) chiamato **key schedule**. Considerando la lunghezza di un blocco della rete di Feistel di B, ad ogni stadio i opera su due semiblocchi L_i e R_i ognuno di lunghezza $\frac{B}{2}$. Nella prima iterazione la concatenazione $L_0||R_0$ dei semiblocchi è proprio il blocco iniziale da cifrare. Il blocco di destra R_i passa inalterato come blocco di sinistra L_{i+1}, mentre il blocco di destra dell'iterazione successiva, R_{i+1} è ottenuto applicando ad R_i una trasformazione $\boldsymbol{F}$ che dipende dalla round key e il cui risultato viene posto in xor bit a bit con il valore di L_i.



$$\begin{cases} L_1 = R_0 \\ R_1 = L_0 \oplus f(R_0, k_1) \end{cases} \cup \begin{cases} L_{i+1} = R_i \\ R_{i+1} = L_i \oplus f(R_i, k_i) \end{cases}$$

Il processo è **reversibile** viste le **proprietà dello xor esclusivo**, visivamente può essere eseguita "rovesciando" tutte le frecce e dunque la direzione input/output dei semiblocchi, ma **non** è necessario. Infatti è sufficente alimentare la stessa rete con input $R_r||L_r$ e naturalmente usando le round key in ordine inverso. Il numero di roun è legato alla diffusione.

La **funzione** $\boldsymbol{F}$ i cui metodi implementativi variano in base alla tipologia di cifrario scelto ha come scopo:

- di inserire la *chiave nel processo* "mischiandola" con i bit del messaggio.

- di assicurare *dipendenza* ***non*** *lineare* fra plaintext e ciphertext, attraverso un **S-Box**.

Nel caso (ad esempio) di **DES** la funzione ***F*** svolge i seguenti passaggi:

1. i 32 bit di R_i vengono espansi a 48 bit tramite la duplicazione di alcuni bit.
2. l'output di questa computazione viene messo in ***xor*** con la *round key* che nel caso di **DES** è a 48 bit.
3. il risultato deve essere riportato a 32 bit, in vista dello ***xor*** finale con il semiblocco L_i, **S-Box**, che in realtà è costituita da 8 tabelle, ciascuna indicizzata da 6 bit e le cui posizioni contengono 4 bit di output.
4. il risultato prodotto dall'S-Box $\Rightarrow 4(bit) * 8(tabelle)$ viene permutato e posto in ***xor*** con il semiblocco L_i e quindi saremo pronti per l'iterazione successiva.

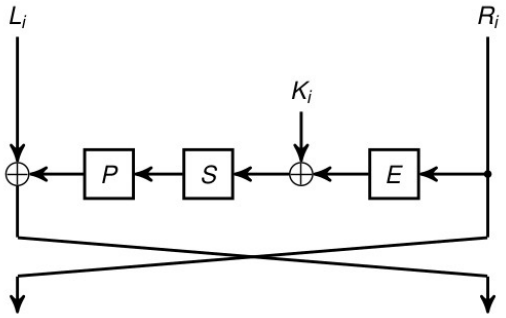


Figura 2.1: F function DES

S_5	4 bit centrali in ingresso															
	0000	**0001**	**0010**	**0011**	**0100**	**0101**	**0110**	**0111**	**1000**	**1001**	**1010**	**1011**	**1100**	**1101**	**1110**	**1111**
bit esterni **00**	0010	1100	0100	0001	0111	1010	1011	0110	1000	0101	0011	1111	1101	0000	1110	1001
bit esterni **01**	1110	1011	0010	1100	0100	0111	1101	0001	0101	0000	1111	1100	0011	1001	1000	0110
bit esterni **10**	0100	0010	0001	1011	1100	1101	0111	1000	1111	1001	1100	0101	0110	0011	0000	1110
bit esterni **11**	1011	1000	1100	0111	0001	1110	0010	1101	0110	1111	0000	1001	1100	0100	0101	0011

Figura 2.2: DES S-Box

Il ***Key Schedule*** è l'algoritmo con cui, dalla chiave iniziale, vengono ottenute le varie ***Round Key***, considerando sempre il caso di **Digital Encryption Standard** osserviamo che presenta una *key* di 56 bit con 8 bit di parità, che viene suddivisa in due metà da 28 bit ciascuna.
Ad ogni round, ciascuna metà viene **ruotata** a sinistra di 1 o 2 posizioni (a seconda del round), dopo di ché viene formata la ***round key*** prelevando 24 bit da entrambe.

2.5 Mode of Operation

Il *mode of operation* è l'algoritmo che permette la cifratura complessiva di un messaggio utilizzando un cifrario a blocchi per le singole porzioni del messaggio. Inanzitutto viene eseguito un riempimento, anche noto come ***padding*** del messaggio iniziale M di modo che la sua lunghezza sia multiplo della dimensione dei blocchi del cifrario selezionato. Indicheremo con $P = P_1||P_2||...||P_n$ la suddivisione dei blocchi dopo il padding.

2.5.1 Electronic Codebooks Mode - ECB

È l'algoritmo più **semplice**, ma anche il più **vulnerabile**. Ogni singolo blocco viene cifrato in manierea *indipendente*, ma attraverso una *stessa chiave*:

$$C_i = E(P_i, k) \quad i = 1, 2, .., n$$

dove indicheremo con ***E*** la funzione realizzata dal cifrario per cifrare il plaintext e k la chiave privata. Il problema del ***ECB*** è che a **blocchi di plaintext identici** corrispondono **blocchi di ciphertext identici**. Ovviamente la problematica è più di rilievo nel caso di immagini o grafici rispetto a messaggi, ad esempio:

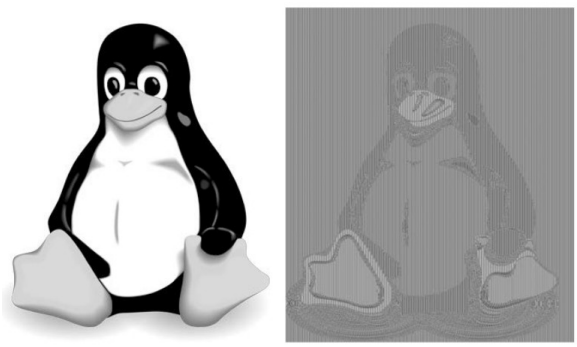


Figura 2.3: Source: J.P. Aumasson, Serious Cryptography, No Starch Press (2018)

2.5.2 Cipher Block Chaining - CBC

In questo differente *mode of operation* la cifratura avviene nel seguente modo:

$$\begin{cases} C_1 = E(P_1 \oplus IV, k) \\ C_i = E(P_i \oplus C_{i-1}, k) \quad i = 2, 3, ..., n \end{cases}$$

dove ***IV*** indica un blocco iniziale, ***Initialization Vector***. La decifrazione è possibile, grazie alle *proprietà dello* ***xor*** *esclusivo*.

$$\begin{cases} P_1 = D(C_1, k) \oplus IV \\ P_i = D(C_i, k) \oplus C_{i-1} \quad i = 2, 3, ..., n \end{cases}$$

Il *vettore di inizializzazione* può essere scelto in vario modo ma la soluzione più comune è di utilizzare un ***nonce***.

ATTENZIONE: è possibile effettuare un attacco di tipo ***Padding Oracle*** se vengono soddisfatte le seguenti condizioni:

1. Lo schema di **padding *ISO/IEC 9797-1*** e con qualsiasi cifrario a blocchi che operi unitamente al **mode of operation *CBC***.

2. *Eve* deve poter effettuare **query di decifrazione** (quindi sarà unicamente necessario che *Eve* conosca **un** ciphertext) ad un server che risponde e **sollevi un eccezione** in caso di ***padding malformato***

Capitolo 3

Aritmetica Modulare

L'*aritmentica modulare* è un settore della matematica che si occupa dello studio di particolari **insiemi numerici** e delle **operazioni** definite su di essi. Gli insiemi oggetto di studio sono sottoinsiemi finiti dei numeri interi, ciascuno definito tramite un intero positivo n, detto **modulo** $\Rightarrow \mathbb{Z}_n$.
$\mathbb{Z} \Rightarrow$ insieme matematico per i numeri interi positivi.
$\mathbb{Z}_n = \{0, 1, ..., n-1\} \Rightarrow$ può essere visto come l'insieme dei ***possibili resti*** quando si divide un numero intero per n.

Note Terminologiche

$y \bmod n = x \bmod n \Rightarrow x$ è congruo $y \bmod n$, anche rappresentato come: $x \equiv y \bmod n$.
$\Rightarrow$ implica che il resto di $\frac{x}{n}$ e $\frac{y}{n}$ siano uguali, avremo quindi che $x - y = kn$.

Operazioni Modulari

Negli insiemi $\mathbb{Z}_n$ sono definite quattro operazioni aritmetiche:

$Op_n = \{+, -, \times, \div\} \Rightarrow$ Nel caso della $\div$ sarà necessaria una condizione affinché sia garantita all'interno di un gruppo $\mathbb{Z}_n$.

Indicheremo con $+_n$ la somma modulare. Ogni operazione coincide con la medesima operazione sugli interi seguita dal calcolo del resto modulo n. Se indichiamo con $\boldsymbol{Q}$ e $\boldsymbol{R}$ rispettivamente il quoziente e il resto della divisione $\frac{a}{b}$ deve valere:

$$a = b \cdot Q + R \rightarrow R \in \{0, 1, ..., b-1\} \quad (0 \leq R < b)$$

Questa condizione deve essere sempre rispettato, anche quando il resto è negativo. Bisogna guardare il rappresentante del gruppo $\mathbb{Z}_n$.

$-1 \bmod N = N - 1 \Rightarrow \mathbb{Z}_n$ è chiuso rispetto alle operazioni definite su di esso.

Elemento Neutro e Inverso in $\mathbb{Z}_n$

Nel gruppo **additivo** ${\mathbb{Z}_n}^+$ avremo:

- 0 è l'elemento **neutro**.
- l'opposto di $x \in {\mathbb{Z}_n}^+$ è l'**inverso additivo** ed è calcolabile come:

$$z \in {\mathbb{Z}_n}^+ \mid x +_n z = 0 \quad \Rightarrow \quad z = n - x$$

Per parlare invece dei due elementi: **neutro** e **inverso** nel gruppo **moltiplicativo** ${\mathbb{Z}_n}^*$ è necessario chiarire la condizione per poter dividere all'interno del nostro gruppo $\mathbb{Z}_n$. Negli insieme $\mathbb{Z}_n$ la divisione non può essere definita a partire dalle corrispondenti operazioni sugli interi, perché

$$\text{in } \mathbb{R} \text{ se } x = \frac{a}{b} \Rightarrow a * \frac{1}{b}$$

Dove $\frac{1}{b}$ è l'**inverso moltiplicativo** di a in $\mathbb{R}$, ma in aritmetica modulare e più precisamente in $\mathbb{Z}_n$ non è sempre definito l'inverso moltiplicativo di b, con $b \in \mathbb{Z}_n$.

Definiamo invece l'**inverso moltiplicativo** di un elemento $x \in \mathbb{Z}_n$ quel numero z, se esiste:

$$x \times_n z = (x * y) \bmod n = 1$$
$$\exists z,\ z \in \mathbb{Z}_n \iff gcd(z, n) = 1$$

Andando a definire che il valore z esiste all'interno del gruppo $\mathbb{Z}_n$ se e solo se l'$\boldsymbol{MCD}(z, n)$ è pari a 1, ovvero se e solo se z e n sono **realtivamente primi** tra di loro (**coprimi**).
$\Rightarrow$ se n è un numero **primo**, allora per ogni elemento appartenente a $\mathbb{Z}_n \backslash \{0\}$ avrà un **inverso moltiplicativo**

$$\text{se } n = p \Rightarrow \forall x \in \mathbb{Z}_p,\ \exists z \in \mathbb{Z}_p \mid (x * z) \bmod p = 1$$

Identità di BEZOUT

Dati due numeri interi x e n, non entrambi nulli, esistono due numeri interi a e b tali che: $gcd(x, n) = a * x + b * n$. Inoltre **MCD**($x$, n) è il più piccolo intero positivo esprimibile come combinazione lineare di x e n.

1. $gcd(x, n) = 1$

2. $1 = a * x + b * n$

3. $a * x = 1 - n * n$

4. $(a * x) \bmod n = 1 - (b * n) \bmod n$

5. $(a * x) \bmod n = 1$

1. Dimostriamo che solo se x e n sono primi allora esiste l'inverso di x

2. Applichiamo l'*identità di bezout*

4. Applichiamo il modulo a destra e a sinistra

5. $(b*n) \bmod n$ è 0 perché $b*n$ è multiplo di n e quindi il resto della divisione per n è 0.

In questo caso vediamo che a ha la stessa proprietà di z ovvero che è l'inverso di $x \in \mathbb{Z}_n$. Avremo quindi che $a \equiv x^{-1} \bmod n$.

Calcoli in $\mathbb{Z}_n$

Supponiamo di dover calcolare un espressione $\epsilon_{\mathbb{Z}_n}$ in $\mathbb{Z}_n$ cioè un espressione in cui tutte le operazioni sono modulari e sia $\epsilon_{\mathbb{Z}}$ la corrispondente espressione in $\mathbb{Z}$ vale:

$$\epsilon_{\mathbb{Z}_n} = \epsilon_{\mathbb{Z}} \bmod n$$

$\Rightarrow$ eseguire il calcolo del resto solo sull'ultimo valore calcolato (vale anche il contrario: è possibile portar eil modulo osu qualsiasi valore intermedio).
Se m divide n allora per qualsiasi valore $x \in \mathbb{Z}_n$ allora $(x \bmod n) \bmod m = x \bmod m$.

Gli **Algoritmi di Euclide** ed **Euclide Esteso**

L'*algoritmo di Euclide Esteso* costituise un metodo **efficente** per calcolare l'*inverso moltiplicativo* di un elemento $x \in \mathbb{Z}_n$ se esiste.
Partiamo, per semplicità enunciando l'*algoritmo di Euclide* che permette di calcolare il massimo comune divisore (***MCD***) di due numeri interi. L'algoritmo è basato sull'ugualianza: $gcd(x, y) = gcd(y, x \bmod y), \quad y \neq 0$ che unito alla "condizione ba-

se" $gcd(x,0) = x$ nel caso in cui y sia nulla permette di **definire per ricorrenza** il teorema:

$$gcd(x,y) = \begin{cases} x & y = 0 \\ gcd(y, x \bmod y) & y \in \mathbb{Z}\backslash\{0\} \end{cases}$$

DIMOSTRAZIONE: dobbiamo dimostrare che le seguenti equazioni siano verificate nello stesso momento: $gcd(x,y) \leq gcd(y, x \bmod y)$ e $gcd(y, x \bmod y) \leq gcd(x,y)$. Consideriamo:

1. se un numero m divide x e y allora divide una qualsiasi **combinazione linerare** a coefficenti interi x e y.

$$m|a \wedge m|b \Rightarrow m|(a * x + b * y) \quad a, b \in \mathbb{Z}$$

2. $x \bmod y$ è una combinazione linerare di x e y.

$$x \bmod y = x - \lfloor \tfrac{x}{y} \rfloor * y = R$$

Poiché $gcd(x,y)$ divide sia x che y, ma anche qualsiasi combinazione lineare a coefficenti interi x e y, quindi $a = 1, b = \lfloor \frac{x}{y} \rfloor$ per cui vale che $a, b \in \mathbb{Z}$. Avremo che il $gcd(x,y)$ divide $x - \lfloor \frac{x}{y} \rfloor * y$ che non è altro che $x \bmod y$, ma divide anche y. Ne consegue:

$$gcd(x,y) \mid x \bmod y$$
$$gcd(y, x \bmod y) \text{ è il valore massimo fra i divisori comuni tra } y \text{ e } x \bmod y$$
$$gcd(y, x \bmod y) \geq gcd(x,y)$$

Poiché $gcd(y, x \bmod y)$ divide sia y che $x \bmod y$ allora divide anche qualsiasi combinazione lineare a coefficenti interi di y e $x \bmod y$. Per questo secondo caso dovremo tenere conto della combinazione lineare: $x = y * Q + R$ dove $Q = \lfloor \frac{x}{y} \rfloor$, ne consegue:

$$gcd(y, x \bmod y) \mid x$$
$$gcd(y, x \bmod y) \mid y * Q + R$$
$$gcd(y, x \bmod y) \mid y * \lfloor \tfrac{x}{y} \rfloor + (x \bmod y)$$

$$x = a * y + b * x \bmod y = a * y + b * (x - \lfloor \frac{x}{y} \rfloor * y)$$
$$= \lfloor \frac{x}{y} \rfloor * y + 1 * (x - \lfloor \frac{x}{y} \rfloor * y)$$

$$gcd(x, y) \geq gcd(y, x \bmod y)$$

Algoritmo di Euclide Esteso

Possiamo modificare la ricorsione del metodo di Euclide in modo da calcolare oltre che $gcd(x, y) = m$ anche i due valori interi tali a e b che soddisfano l'**identità di *Bezout***:

$$m = a * x + b * y$$

1. caso base: $y = 0 \quad m = a * x + 0 = 1 * x = m$, ottenendo, quindi, $a = 1$ e $b = 0$, possiamo quindi definire la formula generale della ricorrenza tramite:

$$EEuclide(x, y) = \begin{cases} x, 1, 0 & y = 0 \\ m, b, a - b * \lfloor \frac{x}{y} \rfloor & m, a, b = EEuclide(y, x \bmod y) \end{cases}$$

DIMOSTRAZIONE: attraverso *ipotesi induttiva*.

$$m = gcd(x, y) = gcd(y, x \bmod y) = a * x + b * y$$
$$= a * y + b * x \bmod y = a * y + b * (x - \lfloor \frac{x}{y} \rfloor)$$
$$= a * y + b * x - b * \lfloor \frac{x}{y} \rfloor * y$$
$$m = b * x + (a - b * \lfloor \frac{x}{y} \rfloor) * y$$

Calcolo dell'inverso

Possiamo utilizzare l'algoritmo di *euclide esteso* per il calcolo dell'inverso moltiplicativo di $x \bmod n$:

$$m, a, b = EEuclide(x, n)$$
$$x^{-1} \bmod n = \begin{cases} 0 & m > 1 \\ a \bmod n & m = 1 \end{cases}$$

Teorema Cinese dei Resti

È lo strumento teorico per cui è possibile ridurre il tempo di computazione nelle operazioni di crifratura (RSA) e decifratura. Definiamo n un numero intero esprimibile come prodotto di $r > 1$ numeri interi tra di loro **coprimi**.

$$n = n_1 * n_2 * ... * n_r$$

Allora il resto della divisione di un qualsiasi numero intero a per n, ovvero $a \bmod n$ è **completamente determinato** dai resti delle divisioni per $n_1, n_2, ..., n_r$. In altri termini esiste una **corrispondenza biunivoca** fra Z_n e il **prodotto cartesiano** $Z_{n_1} \times Z_{n_2} \times ... \times Z_{n_r}$.

$a \in \mathbb{Z}_n \qquad a \to (a_1, a_2, ..., a_r) \to (a \bmod n_1, a \bmod n_2, ..., a \bmod n_r)$

$c = c_1 * a_1 + c_2 * a_2 + ... + c_r * a_r \qquad c_i \in 0, 1 \to C \bmod n_i = a_i$

$m_i = \prod_{j \neq i} n_i \qquad \Rightarrow$ prodotto di tutti i moduli ad eccezione proprio dell'i-esimo n_i

$gcd(m_i, n_i) = 1 \quad \Rightarrow$ per il passaggio di costruzione di m_i

$c_i = m_i * ({m_i}^{-1} \bmod n_i) \Rightarrow c_i \bmod n_j = 0, c_i \bmod n_i = 1$

$C = \sum_{i=1}^{r} c_i * a_i \qquad \Rightarrow c \bmod n_i = a_i, \quad i = 1, ..., r$

$C \bmod n = a \bmod n \Rightarrow C = a$ hanno la stessa rappresenzatione in $\mathbb{Z}_n$

Gruppi e Gruppi Ciclici $\mathbb{Z}_n$ e ${\mathbb{Z}_n}^*$

L'operazione $\{+_n\}$ in $\mathbb{Z}_n$ gode delle seguenti proprietà:

1. **Chiusura**: la somma (modulare) di due elementi in $\mathbb{Z}_n$ è ancora un elemento di $\mathbb{Z}_n$: $z = x +_n y \; \exists z \in \mathbb{Z}_n, \forall x, y \in \mathbb{Z}_n$
2. **Associativa**: $(a +_n b) +_n c = a +_n (b +_n c)$
3. **Esistenza dell'Elemento Neutro**: 0
4. **Esistenza dell'Elemento Inverso** $x \in \mathbb{Z}_n \to x^{-1} = n - x$

In questo modo possiamo dire che $\mathbb{Z}_n$ è un **gruppo** rispetto alla somma modulare. Consideriamo ora l'operazione $\{\times_n\}$ e consideriamo sempre l'insieme $\mathbb{Z}_n$ allora

$\exists\, x^{-1} \in \mathbb{Z}_n \iff gcd(x, n) = 1$.

Se x e n non sono coprimi non esiste l'inverso di x in $\mathbb{Z}_n$ e quindi $\mathbb{Z}_n$ non è un gruppo rispetto alla moltiplicazione (modulare). $\mathbb{Z}_n{}^*$ è il gruppo moltiplicativo in n composto da tutti quegli elementi $x \in \mathbb{Z}_n \mid gcd(x, n) = 1$.

Es.

$\mathbb{Z}_{12}^* \neq \mathbb{Z}_{12} \Rightarrow \{1, 5, 7, 11\} \neq \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$

Se *n* è primo $\mathbb{Z}_n$ è un campo

$\mathbb{Z}_p = \{0, 1, ..., p-1\} \Rightarrow$ conserva tutte le proprietà per essere un gruppo additivo (chiusura, associativa, esistenza dell'elemento neutro e inverso rispetto alla somma).
Siccome p è primo. Presenta tutti gli inversi $\forall x \in \mathbb{Z}_n$, perché $\forall x \in \mathbb{Z}_p \exists x^{-1} \in \mathbb{Z}_p \rightarrow gcd(x, p) = 1 \;\; \forall x \in \mathbb{Z}_p$ (chiusura, proprietà associativa, esistenza dell'elemento neuto, 1, e inverso rispetto alla moltiplicazione).
$\mathbb{Z}_p$ presenta anche la proprietà **distributiva** poiché $\mathbb{Z}_p$ è sia un gruppo *additivo* che *moltiplicativo*. In questo caso $\mathbb{Z}_p$ è un **Campo Finito**, in matematica definiamo *campo* una struttura algebrica composta da un insieme non vuoto e da duue operazioni binarie intere: somma e prodotto. Un **Campo Finito** viene anche chiamato: **Campo di *Galois***.

Ordine di un elemento in un gruppo

Viene definito *ordine* di x appartenente ad un gruppo il numero che rappresenta il minimo numero di volte che dobbiamo fare $\{Op.\}x$ per ottenere l'elemento neutro, partedo dall'elemento neutro stesso.

- **Gruppi Additivi**: $ord(x), x \in \mathbb{Z}_n \Rightarrow x * ord(x) \bmod n = 0$
- **Gruppi Moltiplicativi**: $ord(x), x \in \mathbb{Z}_n{}^* \Rightarrow x^{ord(x)} \bmod n = 1$

Gruppi Ciclici e Generatori

Il numero di elementi di un gruppo G è detto **ordine di *G***, un gruppo si dice **ciclico** se esiste almeno un elemento $g \in G$ che abbia ordine pari all'ordine del gruppo.

$$ord(g) = ord(G)$$

g se è verificata l'ugualianza precedente è detto **generatore** o anche **radice primitiva** di G.
I gruppi additivi $\mathbb{Z}_n$ e moltiplicativi $\mathbb{Z}_n^*$ con n primo sono **gruppi ciclici**, infatti se g è un generatore vale:

$$\mathbb{Z}_n^* = \{g^i \bmod p | i = 1, ..., p-1\}$$

Consideriamo un elemento di h di ordine s con $s < |\mathbb{Z}_n^*|$ (ovvero non un generatore, nel caso di n primo avremo che la **cardinalità** è $p-1$) possiamo comunque considerare l'insieme H generato da h:

$$H = \{h^i \bmod p | i = 1, ..., s\}$$

H è un sottogruppo ciclico di $\mathbb{Z}_p^*$ (ogni sottogruppo deve sempre includere l'elemento 1, infatti è l'elemento neutro di un gruppo moltiplicativo).

$$x = h^i \bmod p \quad i \leq s \Rightarrow s - i \text{ è l'inverso di } i$$
$$x^{-1} \bmod p = h^{s-i} \bmod p \Rightarrow \text{moltiplico per gli inversi}$$
$$x * x^{-1} \bmod p = h^i * h^{s-i} \bmod p \Rightarrow \text{sostituisco}$$
$$h^i * xx^{-1} \bmod p = h^i * h^{s-i} \bmod p$$
$$x^{-1} \bmod p = h^{s-i} \bmod p$$
$$x \text{ ha inverso moltiplicativo } h^{s-i} \bmod p$$

Teorema di Lagrange: Qualsiasi sottogruppo di un gruppo di ordine k ha necessariamente ordine che è un divisore di k.

Teorema Fondamentale dei Gruppi Ciclici: Per ogni divisore k dell'ordine del gruppo esiste uno ed uno solo sottogruppo di ordine k. Ovvero se 2 o più generatori h di sottogruppi di $\mathbb{Z}_p^*$ con uguale ordine (o cardinalità) $\Rightarrow$ allora quei due generatori genereranno lo stesso sottogruppo.

$$H = \{h^i \bmod p | i = \{1, ..., s\}\}$$
$$H' = \{h'^i \bmod p | i = \{1, ..., s\}\}$$
$$ord(H) = ord(H') \Rightarrow H = H'$$

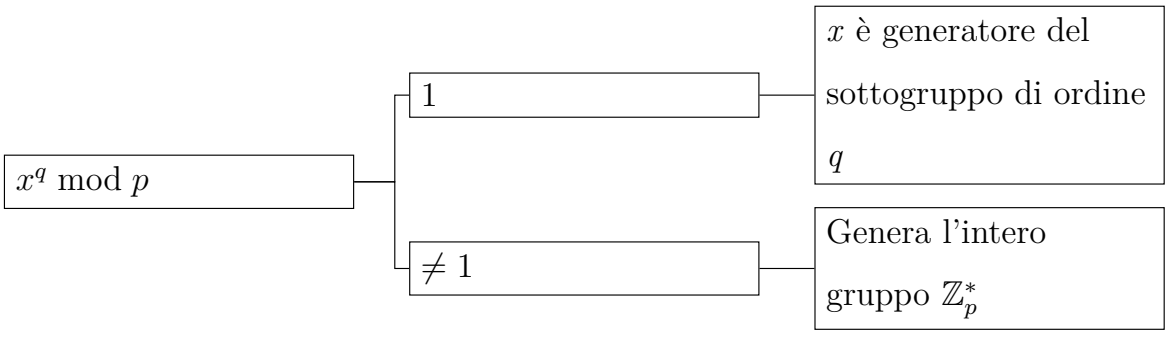
Strong Prime o ***Safe Prime***: i *safe prime* vengono definiti come $p = 2 * q + 1$ dove anche q è un numero primo. Per il teorema di *Lagrange* l'ordine dei sottogruppi non banali può essere o *2* o *q*.

$$ord(\mathbb{Z}_p^*) = p - 1 \quad \Rightarrow \quad p = 2 * q + 1$$
$$p - 1 = 2 * q \quad \Rightarrow \textit{ 2 e } q \text{ sono divisori di } p \text{ - } 1$$

Per il teorema fondamentale dei *gruppi ciclici* ssappiamo che di gruppi di ordine *2* ne esisterà solo *1* e di gruppi di ordine q ne esisterà soltamento *1*. Tralasciando il generatore $h = p - 1$ che genera il sottogruppo $\{-1, 1\}$ che in modulo sarà $\{p - 1\}$ ci saranno metà elementi che genereranno il gruppo di ordine p - *1* e l'altra metà degli elementi che generano il sottogruppo di ordine q.
Nel caso di **strong prime** avremo:

$$x \in H \rightarrow k^2 \bmod p = x \forall k \in \mathbb{Z}_p^*$$
$$(H \subseteq \mathbb{Z}_p^*) \quad ord(\mathbb{Z}_p^*) = (ord(H) + 1) * 2$$

Ovvero è formato dagli elementi che sono, in *aritmentica*, **quadrati perfetti** e che in *aritmentica modulare* sono detti **residi quadratici** (modulo p).
$\Rightarrow$ se p è primo si può dimostrare che ogni quadrato perfetto ha esattamente 2 radici che sono uno l'opposto dell'altro. Consideriamo un $x \in \mathbb{Z}_p^*$ e vogliamo capire se genera il sottogruppo di ordine q o il gruppo di ordine p - *1* allora sarà sufficiente calcolare:



Esponenziale Modulare $\rightarrow$ calcolo in maniera efficente di $a^b \bmod n$ (il problema è la grandezza dei numeri in gioco).

Es.

limitiamo la lunghezza a *128 bit* $\rightarrow \log_2 2^{128^{2^{128}}} = 128^{2^{128}} = 2^{135}$bit $\simeq 10^{40}$cifre decimali

$\Rightarrow$ proprietà dell'aritmetica modulare è quella di ridurro lo spazio necessario.

Il problema ora da risolvere è il tempo di esecuzione infatti non possiamo fare $b-1$ moltiplicazioni modulari perché sarebbe carente in termini di tempistiche.

Prodotto Modulare Essendo l'esponenziale un prodotto introduciamo un algoritmo per il calcolo della quantità $a * b \bmod n \rightarrow$ calcolando $a * b$ come b - *1* addizioni modulari.

Es.

$z = a * b$ con $b = 26$

1. scriviamo b in base *2*.

$$26_{(10)} = 11010_{(2)}$$

2. utilizziamo la proprietà *distributiva*.

$$\begin{aligned} z &= a * b \\ &= a * 26 = a * (1 * 2^4 + 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0 * 2^0) \\ &= a * 2^4 + a * 2^3 + a * 2^1 \end{aligned}$$

3. tutte le quantità $a * 2^k$ possono essere calcolate partendo da:

$$\begin{cases} a_0 = a * 2^0 = a \\ a_{k+1} = 2 * a_k \end{cases}$$

Implementabile tramite l'operazione di *shift*. Utilizziamo sia i k che ci servono $\{1, 3, 4\}$ che quelli che non ci servono $\{0, 2\}$. Poniamo $z = 0$ e ad ogni iterazione sommiamo a_k a z se e solo se il k-esimo bit di b è *1*.

ATTENZIONE: dopo ogni operazione viene eseguita la riduzione in modulo n.

In **Esponenziale Modulare** avremo:

$$Z_0 = 1e \begin{cases} a_0 = a^{2^0} = a \\ a_{k+1} = a_k^2 = (a^{2k})^2 = a^{2*2*k} = a^{2^{k+1}} \end{cases}$$

Ad ogni iterazione moltiplichiamo a_k a Z se e solo se il k-esimo bit di b è uguale a *1* (ovviamente in questo caso si partirà con $Z = 1$).

Il **Logaritmo Discreto** in $\mathbb{Z}_n$ vale $b^e = x \bmod n$, l'esponenziale e è detto logaritmo discreto in base b di x e si scrive $e = \log_b x \bmod n$.
$\Rightarrow$ se n è primo esiste sempre una base g per cui il logaritmo discreto è definito per ogni elemento $a \in \mathbb{Z}_p^*$, tale base deve essere un **generatore** del gruppo.

Logaritmo Discreto Additivo $\mathbb{Z}_n^+$ $\Rightarrow$ ci tornerà comodo nelle curve elittiche.

$$k * g = x \bmod n$$
$$k = \log_g x$$

Radici Primitive (GENERATORI)
Per un gruppo $\mathbb{Z}_p^*$ arbitrario non sono però noti algoritmi polinomiali per decidere se un elemento $a \in \mathbb{Z}_p^*$ è una radice primitiva o meno né tantomeno algoritmi polinomiali per trovarne una. Il numero di radici primitive di $\mathbb{Z}_n^*$ è pari alla funzione **TOZIENTE di EULERO** $\phi(n)$.

$$\phi(n) = |\{i | 1 \leq i < n \wedge gcd(i, n) = 1\}|$$

$\Rightarrow$ $\phi(n)$ è il numero di interi minori di n e coprimi con n.
Il calcolo del logaritmo è apparentemente difficile, infatti non sono noti algoritmi polinomiali per il calcolo del logaritmo discreto $\Rightarrow$ *worst-case* esponenziale, ovvero non fanno meglio dell'approccio a *brute-force* che prova tutti gli esponenti finché non trova quello corretto:

$$g^e \bmod p = x \quad \Rightarrow e = \log_g x \bmod p$$

TEMPO ATTESO è $O(2^{n-1})$ con n il numero di bit

Algoritmo *Baby-Step Giant-Step* (BSGS)

È un algoritmo di tipo *brute-force* che ha un tempo di attesa $O(2^{\frac{n}{2}})$ dove però, entro certi limiti, è possibile privilegiare lo **spazio** o il **tempo**.

CONDIZIONI: noto il modulo p, la base g del logaritmo e x di cui vogliamo calcolare l'esponente e tale che $g^e \bmod p = x$.

ITERAZIONI

1. scegliamo r e s tale che $r * s \geq p$ e notiamo che ogni numero $t \in \mathbb{Z}_p$ può essere rappresentato da:

$$t = i +_p j * r \;\; \forall i = 0, 1, ..., r-1 \;\wedge\; \forall j = 0, 1, ..., s-1$$

2. calcoliamo le 2 successioni:

   - *baby-step*: $g_r = g^i \;\; i = 0, 1, ..., r-1$
   - *giant-step*: $g_s = x * g^{-jr} \;\; j = 0, 1, ..., s-1$

3. siccome possiamo vedere un qualsiasi elemento $t \in \mathbb{Z}_p$ come $t = i + jr$ allora anche l'esponente di $g \bmod p$ sarà incluso, quindi:

$$g^{i+jr} = x \;||\; \text{con } i \text{ e } j \text{ arbitrari}$$
$$g^{i+jr} \bmod p = g^e \bmod p = x \;||\; e = i + jr, \text{ con } e \in \mathbb{Z}_p$$
$$g^i * g^{jr} = x$$
$$g^i * g^{jr} * g^{-jr} = x * g^{-jr}$$
$$\textbf{baby step} \;||\; g^i = x * g^{-jr} \;||\; \textbf{giant step}$$

4. si continuerà ad iterare finché g^i quindi *baby-step* non sarà uguale a $x * g^{-jr}$ ovvero *giant-step*.

Il **tempo di esecuzione** dell'algoritmo è $O(r+s)$ quindi per portare l'utilizzo di tempo minimo bisogna vincolare r e s pari a $r = s = \lceil \sqrt{p} \rceil$, mentre il **consumo di spazio**

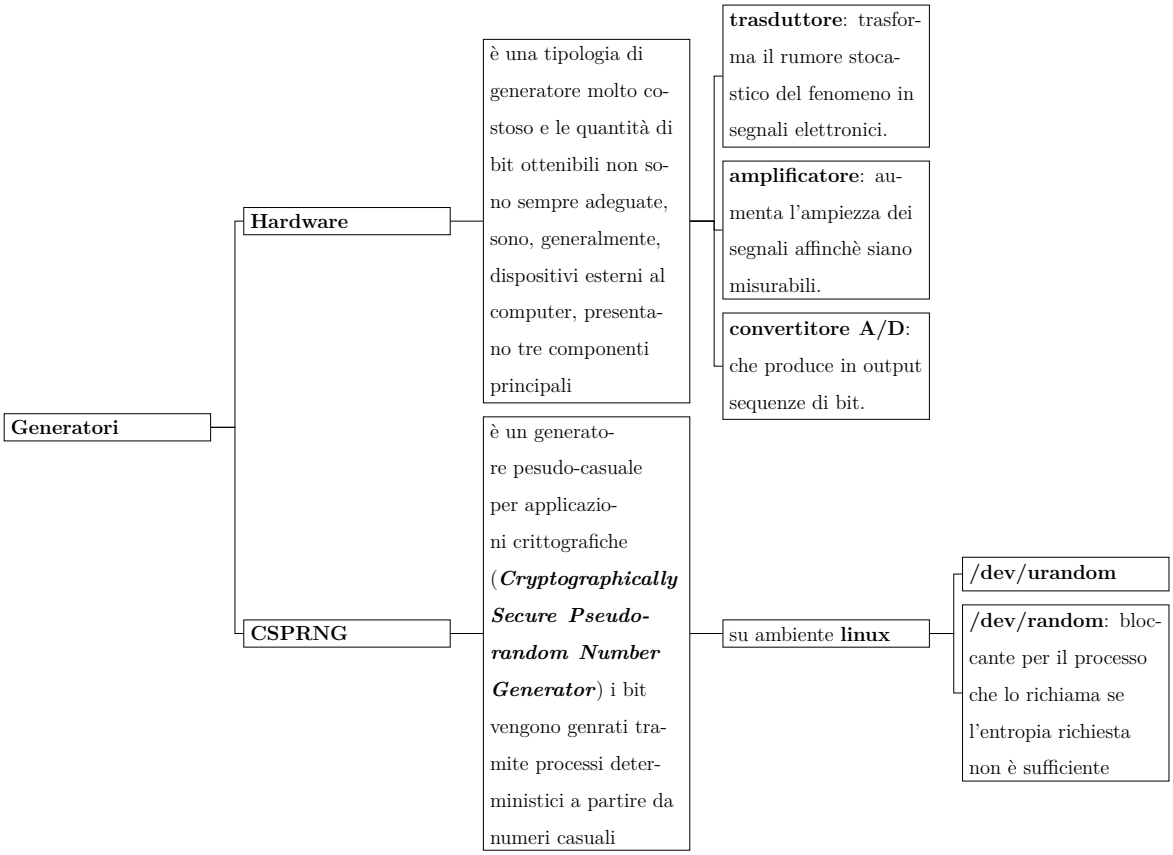
è $O(r)$ e quindi si puòè ridurre in maniera arbitraria al prezzo di aumentare il tempo.

Randomness

In crittografia è una risorsa **indispensabile** avere sequenze di bit generati casualmente indipendenti e con probabilità uniforme o almeno, tendente a.

$$prob[x_i = 0] = prob[x_i = 1] = \frac{1}{2}$$

Attraverso una qualche sorgente di casualità che sarà in grado di produrre eventi casuali in un opportuno spazio.



3.1 Algoritmi Probabilistici

Un algoritmo viene definito probabilistico se utilizza una qualche sorgente di casualità, sono normalmente molto utilizzati in crittografia asimmetrica (non vengono utilizzati *pseudo-random* visto che la robustezza dei cifrari asimmetrici si basa sulla **teoria dei numeri**).
Valutazione: ogni operazione aritmentica/logica ha un peso che dipende dal numero di bit degli operandi (*MODELLO*: bit cost).
Gli **algoritmi decisionali** sono algoritmi probabilistici che però hanno come outpu un valore binario [0 | 1], in questo caso l'elemnto di casualità, nel comportamento può risiedere nel risultato o nel tempo di calcolo.

3.1.1 Algoritmi di Monte Carlo

Un algoritmo proabilistico di tipo **Monte Carlo** per un problema decisionale P, ha le seguenti caratterisitiche:

1. su input la cui risposta esatta è *1* restituisce *1* con un a probabilità fissa $\epsilon > 0$.
2. su input $x \in P$ la cui risposta esatta è *0*, restituisce *0*.

Probability-Bounded One-Side Error Monte Carlo

1. ****probability-bounded (away from 0)***: la probabilità di errare non solo deve essere positiva, ma che tenda a *0* al crescere della dimensione dell'input.
2. ***one-side error***: perché l'algoritmo può sbagliare solo quando risponde *false* e la risposta esatta è *true*.

[*] è necessario il ***probability-bounded*** affinché si possa identificare un numero n tale per cui si raggiunge una probabilità accettabile per dire che la risposta ottenuta è quella corretta.
$\boldsymbol{N}$ è il numero di *run* indipendenti tali che al crescere di N cresce la probabilità di

rispondere correttamente ("indovinare la classe di appartenenza").

Es. $\epsilon > 0,001$

$\Rightarrow$ la probabilità che restituisco n volte *false* è limitata superiormente dal prodotto delle singolo probabilità.

$$\begin{aligned} P[true] &= (1-0,01)^n = 0,99^n \\ &= (1-0,01)^{69} = 0,99^{69} \\ &= 0,4998 \end{aligned}$$

con n maggiori è possibile andare ad aumentare la probabilità di successo $\to 1$.

3.1.2 Algoritmi di Las Vegas

Un algoritmo probabilistico di tipo **Las Vegas** restituisce sempre la risposta corretta in tempo determnato solo con una data probabilità.

Es. **Procedura di Bit Uniformi** $p(x_i = 0) = p(x_i = 1)$ e indipendenti da una sorgente di bit indipendenti ma non uniforme $p(z_i = 0) = p \wedge p(z_i = 1) = 1 - p$ con un valore $0 < p < 1$.

$$\begin{aligned} p(z_i, z_{i+1}) = (0,0) \quad || \quad p^2 \\ p(z_i, z_{i+1}) = (0,1) \quad || \quad p \cdot (1-p)2 \\ p(z_i, z_{i+1}) = (1,0) \quad || \quad p \cdot (1-p) \\ p(z_i, z_{i+1}) = (1,1) \quad || \quad (1-p)^2 \end{aligned}$$

Finché le coppie di z_i e z_{i+1} sono uguali l'algoritmo non restituisce nulla. $S = 1 - 2 \cdot p \cdot (1-p)$ è la probabilità che z_i e z_{i+1} siano uguali. Consideriamo che la probabilità per la quale al k-esimo tentaivo venga fornito un bit uniforme.

$$\begin{gathered} s^{k-1} \cdot (1-s) \Rightarrow \text{Distribuzione Geometrica} \\ \textstyle\sum_{k=1}^{\infty} k \cdot s^{k-1} = \frac{1}{1-s} \quad \alpha \quad p \end{gathered}$$

Possiamo anche analizzare la relazione fra i bit casuali e la probabilità che l'algoritmo termini entro un certo numero di tentativi:

$$\Rightarrow \text{probabilità che } 2 \cdot k \text{ bit siano sufficienti}$$
$$\sum_{i=1} k s^{i-1} \cdot (1-2) = 1 - s^k$$

3.2 Teoria e Algoritmi per il Test di Primalità

$\pi(n) \Rightarrow$ i primi non maggiori di n.

$\pi(n) \simeq \frac{n}{\log_2 n}$ $\quad || \Leftarrow$ ***Prime Number Theorem***

$\Rightarrow$ al crescere di n, nei primi n numeri c'è una probabilità $\frac{1}{\log_2 n}$ di avere un primo.

Per avere un primo a n bit dobbiamo effettuare mediamente $n \cdot \log_2 n \approx 0.69 \cdot n$ tentativi.

Piccolo Teorema di Fermat

Se p è primo allora vale $x^{p-1} \bmod p = 1 \ \ \forall x \in \mathbb{Z}_p \mid gcd(x,p) = 1$

$\Rightarrow$ non può funzionare come test di primalità perché se $gcd(x,n) = 1$ non per forza n è prima, ma potrebbe essere un *pseudo-primo* base x (con $x = 2$ tra numeri a 512 bit avremo la possibiltà di 1 su 10^{20} che n sia primo).

$a \in \mathbb{Z}_n$ è detto **residuo quadratico** (modulo n) $\rightarrow \exists x \in \mathbb{Z}_n \mid a = x^2 \bmod n$

Alcune proprietà dei **R.Q.**:

- se a è un $R.Q.$ allora anche $a^{-1} \bmod n$ è un $R.Q.$.
- l'insieme dei $R.Q.$ formano un sottogruppo di $\mathbb{Z}_n^* \rightarrow Q_n \subseteq \mathbb{Z}_n^*$.
- n è primo $\Rightarrow card(\mathbb{Z}_p^*) = p - 1 \Rightarrow card(Q_n) = \frac{card(\mathbb{Z}_p^*)}{2} = \frac{p-1}{2}$ ed è formato da tutte le potenze di un generatore g.

Simobolo di Legendre

$$\Rightarrow \left(\frac{a}{p}\right) = \begin{cases} 0 & \text{se } a \equiv 0 \bmod p, \text{ se p divide a} \\ 1 & \text{se } a \text{ è un R.Q. mod } p \\ -1 & \text{se } a \text{ non è un R.Q. mod } p \end{cases}$$

Per calcolare il *Simbolo di Legendre* utilizziamo il **Criterio di Eulero** che enuncia:

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

Proprietà del **Simbolo di Legendre**:

- $(\frac{ab}{p}) = (\frac{a}{p}) \cdot (\frac{b}{p})$
- se $a \equiv b \bmod p \Rightarrow (\frac{a}{p}) = (\frac{b}{p})$
- **reciprocità quadratica** se p è primo q è primo $\Rightarrow (\frac{q}{p}) \cdot (\frac{p}{q}) = (-1)^{\frac{(p-1)(q-1)}{4}}$
- calcolare se 2 è un $R.Q. \Rightarrow (\frac{2}{p}) = (-1)^{\frac{p^2-1}{8}}$ ovvero 2 è un $R.Q.$ se $p \equiv 1, 7 \bmod 8$
- calcolare se 5 è un $R.Q. \Rightarrow (\frac{5}{p}) = (-1)^{\frac{2p+2}{5}}$ ovvero se $p \equiv 1, 4 \bmod 5$

Simobolo di Jacobi È una generalizzazione del **Simbolo di Legendre** infatti n è un numero composto fattorizzabile in $n = p_1 \cdot p_2 \cdot ... \cdot p_k$ con p_i primo, nel caso in cui n sia un numero primo allora il **Simbolo di Jacobi** è uguale al **Simbolo di Legendre**.

$$(\tfrac{a}{n}) = \textstyle\prod_{i=1}^{k} (\tfrac{a}{p_i}) = \begin{cases} 0 & \text{se } a \equiv 0 \bmod n \\ 1 & \text{se } a \not\equiv 0 \bmod n \cup a \equiv x^2 \bmod p \Rightarrow \text{ può essere un R.Q.} \\ -1 & \text{se } a \text{ non è un R.Q.} \end{cases}$$

Proprietà del **Simbolo di Jacobi**:

- **moltiplicativa**: $(\frac{ab}{n}) = (\frac{a}{n}) \cdot (\frac{b}{n})$
- se $a \equiv b \bmod n \Rightarrow (\frac{a}{n}) = (\frac{b}{n})$
- se m e n sono interi dispari

$$(\tfrac{m}{n}) = \begin{cases} -(\frac{n}{m}) & \text{se } n, m \equiv 3 \bmod 4 \\ \frac{n}{m} & \text{altrimenti} \end{cases}$$

- $(\frac{2}{n}) = 1 \iff n \equiv 1, 7 \bmod 8$

Tuttavia il **Simbolo di Jacobi** non può essere preso come criterio per stabilire se un numero a è un **residuo quadratico** modulo n.

3.3 Test di Primalità di Solovay - Strassen

$\Rightarrow$ se n è un numero primo sappiamo che vale:

$$(\frac{a}{n}) = \quad a^{\frac{n-1}{2}} \bmod n$$

$$\text{Jacobi} = \qquad \text{Legendre}$$

Se l'ugualianza **non vale** sappiamo per certo che n è composto, mentre se l'ugualianza **vale** non abbiamo la certezza che n sia primo infatti può essere primo oppure pseudo primo di Eulero rispetto alla base a.
Un valore di a per cui $gcd(a,n) > 1 \cup (\frac{a}{n}) \neq a^{\frac{n-1}{2}} \bmod n$ viene chiamato testimone della non primalità di n.

- generaimo un $n > 2$ dispari [*]
  - scegliamo un $a \in \{1, 2, ..., n-1\}$ uniformamente a caso [*]
  - calcoliamo $m = gcd(a,n) \rightarrow m > 1$ ripartiamo [*]
  - calcoliamo $J(a,n) = (\frac{a}{n})$ e $L = a^{\frac{n-1}{2}} \bmod n$
  - se $L \neq J$ ripartiamo [*]
  - se $L = J$ ripartiamo [*]

$\Rightarrow$ L'algoritmo restituisce probabilmente primo se la risposta **corretta** è ***n* primo**.
$\Rightarrow$ L'algoritmo può restituire probabilmente primo se la risposta **sbagliata**: ***n*** composto o pseudo-primo di Eulero rispetto alla base a.

$\forall n$ [composto] $\exists \lceil \frac{n-1}{2} \rceil$ interi a tali che n è **pseudo-primo** e quindi esisteranno anche $\frac{n-1}{2}$ [a] che sono testimoni di non primalità.

il ciclo interno dell'algoritmo erra con probabilità non superiore a $\frac{1}{2} \Rightarrow 50\%$, è quindi possibile aumentare il numero di iterazioni interne per ridurra la probabilità di errore.

3.4 Algoritmo di Fattorizzazione ρ di Pollard

L'algoritmo produce, a partire da un valore $x_1 \in \mathbb{Z}_n$ (normalmente $x_1 = 2$), una successione di valori $x_{j+1} = f(x_j) \bmod n,\ j = 1, 2, ...$ che, almeno idealmente, dovrebbe esibire la proprietà di generare una sequenza casuale, ad un certo punto inizierà a ripetersi (siamo in un gruppo ciclico).

"Ogni tanto", un valore calcolato viene salvato e utilizzato per le iterazioni successive, a caso di studio, vengono salvati i valori $x_j \mid j = 2^i$.

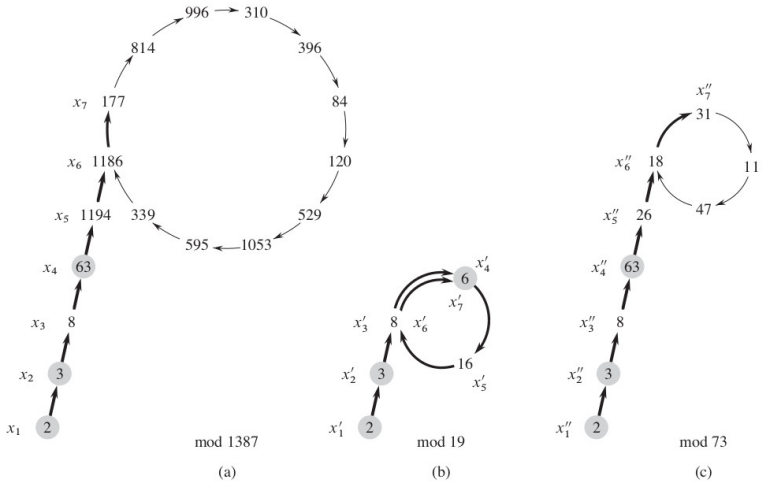
Andiamo a definire come $f(x) = (x^2 - 1) \bmod n$

```
1   j = 1; i = 2; y = x_1 = 2
2   while true:
3       x_{j+1} = (x_j^2 - 1) mod n
4       j += 1
5       m = gcd(y - x_j, n)
6       if m > 1:
7           return m
8       if j == i:
9           i *= 2
10          y = x_j
```

L'algoritmo termina restituisce un fattore non banale di n, se il $gcd(|y - x_j|, n) > 1$ ovvero se $y - x_j \equiv 0 \bmod p$ dove p è un fattore di n in questo modo possiamo vedere $y - x_j = k \cdot p \rightarrow gcd(k \cdot p, n) = p$. Quindi lo scopo dell'algoritmo è quello di andare a cercare coppie di iterate che siano diverse ma **congrue modulo un divisore di n**. La parte "geniale" dell'algoritmo è la strategia con cui viene effettuata la ricerca, ovvero tenendo fissa una delle due iterate nel confronto (la y del pseudo-codice) per periodi via via più lunghi. Per ogni possibile fattore p di n esiste una "successione ombra" $\{z_i\}_{i \geq 1}$, ottenuta riducendo gli x_i modulo il fattore p. Concentrandoci sulla successione per un qualche fattore p di n, possiamo subito dimostrare che la successione ombra ha la **stessa struttura** della $\{x_i\}_{i \geq 1}$

$$\begin{aligned} z_{i+1} &= x_{i+1} \bmod q \\ &= ((x_i^2 - 1) \bmod n) \bmod q \\ &= (x_i^2 - 1) \bmod q \\ &= ((x_i \bmod q)^2 - 1) \bmod q \\ &= (z_i^2 - 1) \bmod q \end{aligned}$$

Ogni successione è descritta da: un "antiperiodo" (il **gambo** del ρ) che ha lunghezza $\lambda = \mathcal{O}(\sqrt{q})$ e un "periodo" (detta **testa** del rho) che ha lunghezza $\sigma = \mathcal{O}(\sqrt{q})$.



Potrebbe succedere che i valori λ e σ per più di una "sequenza ombra" siano tali che la "scoperta" di due iterate identiche avvenga nello stesso istante, se $n = p \cdot q$ con p e q primi fa si che le corrispondenti iterate x_j e x_{2^i} siano congruenti modulo p e modulo q. Questo implica che x_j e x_{2^i} sono anche congrui modulo n e dunque $gcd(x_j, x_{2^i}) = n$ questo ovviamente **non aiuta**. Nel caso di scomposizione $n = p^k$ può risutare semplicemnte che $z_{2^i} = z_j \Rightarrow x_{2^i} = x_j$ e anche in questo caso non si scopre nulla. L'efficienza è data da $p < \sqrt{n} \Rightarrow \mathcal{O}(\sqrt[4]{n})$ operazioni su numeri di $\mathcal{O}(\log n)$ bit.

Capitolo 4

Crittografia Asimmetrica e Scambio di Chiavi

4.1 Diffie-Hellman

Per ovviare al problema dei cifrari simmetrici per quanto riguarda la *gestione delle chiavi* nel 1976 viene pubblicato un articolo *"New Direction in Cryptography"*, pubblicato da due ricercatori: Whitfield Diffie e Martin Hellman. La loro intuizione fu quella di poter utilizzare nella cifrazione e nella corrispondente decifratura due chiavi differenti. Se si potesse allora si potrebbe **condividere** la **chiave di cifratura** e quindi utilizzabile da chiunque mentre la **chiave di decifrazione** tenuta **privata**. Lo schema introduce il "problema" della gestione delle chiavi pubbliche che però non è totalmente risolvibile a livello algoritmico.
Il **protocollo** introdotto da *DH* non è uno schema a chiave pubblica, ma un metodo per la **comunicazione di informazione segreta su un canale di comunicazione insicuro**. Questo protocollo viene tutt'ora utilizzato all'interno di molti altri protocolli di sicurezza su internet, tra cui: *TLS* e *SSH*.

Il protocollo DH su gruppi ciclici $\mathbb{Z}_p^*$
In questo protocolo *Alice* e *Bob* utilizzano un **protocollo simmetrico** per comunicare, per farlo bisogna che però si scambino una chiave condivisa da tenere segreta tramite un canale insicuro. Il protocollo di Diffie e Hellman consente alle due parti di

concordare un segreto che altro non è che un elemento di un oppportuno gruppo $\mathbb{Z}_p^*$. Utilizzando tale segreto, le due parti **deriveranno la chiave** da usare nel protocollo.

Algoritmo DH

1. *Alice* e *Bob* si accordano sul **modulo p** da utilizzare, deve essere un numero primo, e sull'uso di una **radice primitiva** $g \in \mathbb{Z}_p^*$.
   - tutto questo viene fatto **pubblicamente** e quindi anche un malvivente *Eve* li conosce.
   - p e g vengono scelta da chi inizia la comunicazione e confermati dall'altra parte.
2. *Alice* sceglie con **distribuzione di probabilità uniforme** un numero $a \in \mathbb{Z}_p^*$, con il quale calcola $x_a = g^a \bmod p$ e invia x_a a *Bob*.
3. in modo simile *Bob* sceglie un elemento $b \in \mathbb{Z}_p^*$ e con questo valore calcola $x_b = g^b \bmod p$ e invia x_b ad *Alice*.
4. con l'informazione x_a inviata da *Alice*, *Bob* si calcola il valore $z_a = x_a^b \bmod p$.
5. analogamente *Alice* si calcola il valore $z_b = x_b^a \bmod p$ con il valore x_b inviato da *Bob*.
6. la quantità $z = z_a = z_b$ è il **segreto condiviso**.

È possibile verificare l'ugualianza $z_a = z_b$:

$$\begin{aligned} z_a &= x_b^a \bmod p \\ &= (g^b \bmod p)^a \bmod p \\ &= (g^b)^a \bmod p \\ &= (g^a)^b \bmod p \\ &= (g^a \bmod p)^b \bmod p \\ &= x_a^b \bmod p \\ &= z_b \end{aligned}$$

L'**Efficenza** del protocollo è dovuto al fatto che tutte le quantità necessarie sono facilemnte calcolabili in maniera efficente, le computazioni più onerose sono i due esponenziali modulari che hanno **complessità polinomiale** sulla base della lunghezza dei valori (in bit). La vera computazione è nella ricerca di un primo "adatto", infatti bisogna trovare un p primo che abbia come **generatore** un valore g che abbia ordine $ord(g) = p - 1$. La ricerca di un primo "adatto" avviene tramite un ***safe prime*** con generatore **"fisso"**.
La **Sicurezza** si basa sulla difficoltà algoritmica da parte di *Eve* di calcolare z dalle informazioni girate in maniera insicura ovvero: il primo p il generatore g e le due chiavi pubbliche x_a e x_b. *Eve* può effettuare tutte le operazioni che vuole su $\mathbb{Z}_p^*$ ma *Eve* necessiterebbe di una delle due chiavi private o a o di b e questo richiederebbe il calcolo del **logaritmo discreto** per invertire x_a e x_b infatti $x_a = \log_g x_a \bmod p$. Ovviamente la situazione è la stessa se *Eve* vuole reversare la *chiave segreta z*.

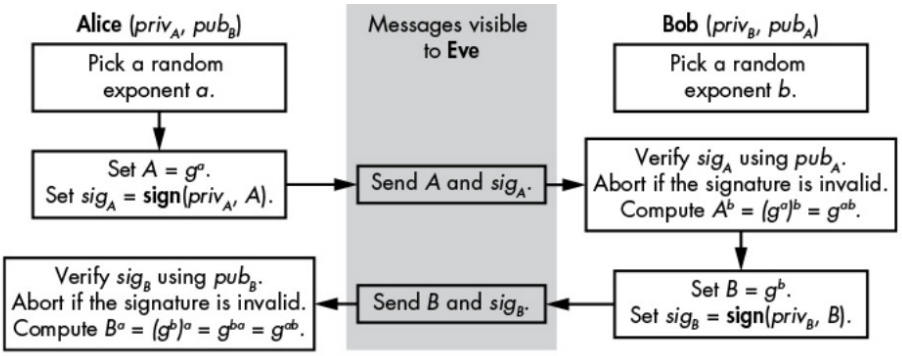
Computational Diffie-Hellman Assumptions se g, a e b sono scelti a caso in $\mathbb{Z}_p^*$ allora, a partire da g, $g^a \bmod p$ e $g^b \bmod p$ la determinazione di $g^{ab} \bmod p$ è un problema **computazione intrattabile**.
Il dato condiviso da *Alice* e *Bob* non è generalmente utilizzabile come chiave segreta in un protocollo di crittografia simmetrica, infatti le chiavi che cerchiamo sono stringhe di bit che hanno, per singolo bit, la stessa probabilità di essere o *0* o *1*. Nel nostro caso il segreto condiviso invece è un numero uniformemente distribuito in $\mathbb{Z}_p^*$ che ovviamente non è la stessa cosa. Infatti in un caso di $\mathbb{Z}_{11}^*$ avremo che la possibilità del bit di essere *1* in posizione 2^2 sarà lo 0.2% a differenza dei bit in posizione 2^0 e 2^1 che invece avranno probabilità di essere *1* pari a quella di essere *0* ovvero 0.5%. Questo devia molto dalla situazione ideale che vorremmo ottenere per una chiave privata in un protocollo simmetrico. Per riuscire ad estrarre **entropia** dal segreto viene applicato al segreto condiviso una funzione **hash crittografica** (identica tra *Alice* e *Bob*) il cui output viene usato come chiave simmetrica.

Attacchi a DH: visto la *computational diffie-hellman assumptions* attacchi diretti per determinare $g^{ab} \bmod p$ sono **computazionalmente infattibili**. Un attacco vincente è invece il **Man-in-the-Middle**. Nel caso in cui *Eve* si interponga tra *Alice*

e *Bob* e si riesca a fingere *Alice* rispetto a *Bob* e viceversa, potrà partecipare ad un doppio scambio di chiavi, questo è possibile in quanto *diffie-hellman* non fornisce un protocollo sicuro di autenticazione. A seconda degli obiettivi, *Eve*, potrà modificiare i messaggi (attacco a **breve termine**) o lasciarli inalterati (attacco a **lungo termine**).

Authenticated Diffie-Hellman

È chiaro che qual'ora *Alice* e *Bob* riuscissero ad **autenticarsi** l'uno rispetto all'altro e viceversa, un attacco **man-in-the-middle** sarebbe irrealistico in quanto, prendendolo come esempio, *Alice* riuscirebbe a svelare l'impersonificazione di *Eve*. Per poter riuscire ad autenticarsi è quindi necessario ricorrere alla **firma digitale**, ovvero entrambi le parti deve possedere anticipatamente una chiave pubblica.



The authenticated Diffie–Hellman protocol

In generale in un protocollo per lo scambio sicuro di chiavi, c'è la necessità che almeno una delle due parti disponga di un **long-term secret** come può essere uno chiave **RSA**, ma a questo punto perché non usare direttamente una crittografia asimmetrica invece che un protocollo di scambio chiavi. Questo tipo di approccio **non** garantisce ***forward-secrecy***.

La ***Forward Secrecy*** è una proprietà dei protocolli di *negoziazione delle chiavi* che assicura che se una chiave di cifratura a lungo termine (*long-term secret*) viene compromessa, le chiavi di sessione generate a partire da essa rimangono riservate.

Ephemeral Diffie-Hellman Keys

In ***TLS 1.3*** le chiavi simmetriche sono invece generatea partire da *chiavi DH effimere*, questo implica che la chiave pubblica e privata dopo essere usate una sola volta, vengono **scartate da client e server**.

4.2 ElGamal

Nel 1985 ***ElGamal*** propone un sistema crittografico a chiave pubblica, questo viene pubblicato qualche anno dopo **RSA** (1977), ma lo affronteremo prima per una similitudine con *Diffie-Hellman*. Poiché non venne brevettato, il sistema crittografico di *ElGamal* venne inserito, insieme al protocollo **DSA** (*Digital Signature Algorithm*), in molte suite crittografiche, tra tutte: ***GNUPG***. All'interno di *ElGamal* possiamo identificare, come in qualunque altro algoritmo di crittografia asimmetrica due elementi:

1. **generazione di una coppia di chiavi**, pubblica (che può essere resa disponibile) e privata.
2. processo di comunicazione, composto a sua volta dalla **cifratura** e **decifratura**.

Per la parte di generazione della coppia di chiavi, inizialmente, *Alice* determina i *parametri* del protocollo:

- un *numero* $\boldsymbol{p}$ *primo* di lunghezza appropriata.
- una *radice primitiva* $\boldsymbol{g}$ di $\mathbb{Z}_p^*$.
- calcolare il valore $A = g^a \bmod p$ dove $a \in \mathbb{Z}_p^*$ è un numero scelto uniformemente a caso.

Una volta determinati i paramentri *Alice* conserverà a come **chiave segreta** e provvederà a diffondere la terna *(p, g, A)* come corrispondente **chiave pubblica**. Come possiamo notare, per questa prima parte del protocollo la differenza con *DH* è solo nei destinatari della comunicazione, infatti non è più solo *Bob* ma tutti. Nel momento in cui *Bob* vuole inviare un messaggio cifrato ad *Alice*, *Bob* ottiene la chiave pubblica di *Alice*, ovvero la terna *(p, g, A)* e, dopo aver scelto un valore $b \in \mathbb{Z}_p^*$ uniformemente a caso, si calcola due quantità:

1. $B = g^b \bmod p$
2. $Cipher = (A^b * M) \bmod p$ dove M è il messaggio che *Bob* vuole inviare ad *Alice*.

Inviando ad *Alice* la coppia $C = (B, c)$ che costituisce il messaggio cifrato. Una volta ottenuto la coppia (B, c) che costituisce il messaggio cifrato *Alice* potrà decifrare il messaggio calcolando per prima cosa $Z = B^a \bmod p$ riuscendo ad ottenere attraverso l'uso del *teorema di Euclide Esteso* il valore $Z^{-1} \bmod p$ e riuscendo a riottenere il messaggio in chiaro $M = (Z^{-1} * c) \bmod p$, si noti che il messaggio deve essere interpretabile come un numero **minore di p**. La **correttezza** dell'algoritmo è conseguenza del fatto che $Z = B^a \bmod p = A^b \bmod p$, ed è esattamente la stessa dimostrazione avvenuta per il teorema di *diffie-hellman*.

L'**Efficenza** è dovuta che la complessità della cifratura è dominata da due calcoli di potenze modulari, a cui va aggiunto il prodotto, mentre per il resto si tratta di pre-computazione delle chiavi. Al contrario la decifratura richiede il calcolo di una potenza, di un inverso e di una moltiplicazione. Un numero sostanzialmente molto ridotto di operazioni onerose che però bisogna considerare su numeri di oltre 1000 bit. Nel caso in cui il messaggio fosse più lungo della lunghezza del modulo p, *Bob* dovrebbe spezzare il messaggio in blocchi e ripetere la cifratura usando un valore di k diverso per ogni singolo blocco. Con molti blocchi, e quindi con messaggi molto lunghi, cifratura e decifratura asimmetrica diventano dei passaggi molto onerosi soprattuto se confrontati con processi di cifratura simmetrica che oggigiorno supportano anche accelleratori hardware. Per questa ragione, la crittografia asimmetrica viene utilizzata in sinergia a quella simmetrica, in particolare, protocolli *asimmetrici* vengono utilizzati in fase di *autenticazione* delle parti e per lo *scambio di chiavi*, mentre per la *comunicazione* vera e propria viene utilizzato un protocollo *simmetrico*.
Andando ad osservare invece la **Sicurezza** di *ElGamal*, questa si basa sulla *Computational Diffie-Hellman Assumptions* (**CDHA**) ovvero girando il problema che supponendo di riuscire a rompere *ElGamal* e quindi di essere in grado di risolvere l'assunzione ***CDH***, infatti *ElGamal* "oscura" il messaggio moltiplicando $(\times_p)$ proprio per una quantità $A^b \bmod p$ che corrisponde al **segreto condiviso di DH**. Se dunque siamo in grado di decifrare M (senza utilizzare il calcolo diretto del logaritmo discreto $a = \log_g M \bmod p$) allora possiamo risalire alla quantità $A^b \bmod p = g^{ab} \bmod p$ ovvero proprio la quantità che la ***CDH Assumptions*** chiede di calcolare. Se quindi vale l'ipotesi *CDHA* possiamo dedurre che **ElGamal è SICURO**.

Un errore da non commettere è di utilizzare per cifrature differenti lo stesso valore b, infatti se questo dovesse valere, allora anche B non cambia, quindi le cifrature di M_1 e M_2 risulterebbero:

$$C_1 = (B, c_1 = (A^b * M_1) \bmod p) \qquad \text{e} \qquad C_2 = (B, c_2 = (A^b * M_2) \bmod p)$$

Consideriamo ora C_1

$$\begin{aligned} C_1 &= (B, c_2) \\ &= (g^b \bmod p,\ A^b * M_1 \bmod p) \\ &= (g^b \bmod p,\ g^{ab} * M_1 \bmod p) \\ &= (g^b \bmod p,\ Z * M_1 \bmod p) \qquad Z = B^a \bmod p = g^{ab} \bmod p \end{aligned}$$

$$c_2 = Z * M_2 \bmod p \qquad M_2 = Z^{-1} * M_2 \bmod p$$

Moltiplichiamo a destra e a sinistra per c_1^{-1}

$$\begin{aligned} c_1^{-1} * c_2 \bmod p &= c_1^{-1} * (Z * M_2) \bmod p \\ &= (A^b * M_1)^{-1} * (Z * M_2) \bmod p \\ &= (g^{ab} * M_1)^{-1} * (Z * M_2) \bmod p \\ &= (Z * M_1)^{-1} * (Z * M_2) \bmod p \\ &= Z^{-1} * Z * (M_1^{-1} * M_2) \bmod p \\ &= (M_1^{-1} * M_2) \bmod p \end{aligned}$$

Moltiplichiamo a destra e a sinistra per M_1

$$M_2 = M_1 * (c_1^{-1} * c_2) \bmod p$$

Se *Eve* fosse in grado (per qualsiasi motivo) di decifrare il messaggio M_1, e quindi avere la coppia $(M_1,\ c_1)$ potrebbe decifrare anche i successivi messaggi cifrati con lo stesso valore b.

4.3 Sistema Crittografico RSA

Il sistema crittografico **RSA** prende il nome dalle iniziali dei loro inventori: *Ronald L. Rivest*, *Adi Shamir* e *Leonard M. Adlemam*. *RSA* basa la sua efficacia, in termine di **sicurezza**, sulla difficoltà computazionale del **problema della fattorizzazione di numeri interi**, ovvero sul fatto che ad oggi non esistono algoritmi efficienti per fattorizzare numeri di grandi dimensioni. Ad oggi, però, non esiste un argomentazione matematica, come nel caso di *Diffie-Hellman*, che dimostri che saper violare efficientemente RSA implichi saper fattorizzare numeri interi.
Warning: un **computer quantistico** sarebbe in grado di fattorizzare in maniera efficacie i numeri interi e quindi violare RSA.

L'**Algoritmo in Dettaglio**: come in ogni cifrario asimmetrico la generazione delle chiavi è eseguita dal destinatario del ciphertext, che identificheremo come *Alice*, il protocollo ha un solo parametro di input, che è la lunghezza in bit, indicata con N, dell'intero con cui si effettuano le riduzioni modulari.

- **Generazione delle Chiavi: *Alice***
  1. generare due primi a casa p e q di lunghezza $\frac{N}{2}$.
  2. calcolare il prodotto $n = p * q$.
  3. calcolare $\phi(n) = (p-1) * (q-1)$.
  4. determinare un intero e, (normalmente $e = 65537$) e relativamente primo con $\phi(n)$ ovvero che $gcd(e, \phi(n)) = 1$.
  5. calcolare l'inverso moltiplicativo d di e modulo $\phi(n)$, che esiste perché e è **coprimo** con $\phi(n)$.
  6. diffondere la coppia ($\boldsymbol{e}$, $\boldsymbol{n}$) come **chiave pubblica** e conservare $\boldsymbol{b}$ come **chiave privata**.
- **Cifratura del Messaggio M: *Bob***
  1. si procura la chiave pubblica di *Alice*, ($\boldsymbol{e}$, $\boldsymbol{n}$).
  2. calcola $C = M^e \bmod n$.

3. invia C come messaggio cifrato ad *Alice*.

- **Decifrazione del Messaggio C: *Alice*** calcola $M = C^d \bmod n$.

Correttezza del Protocollo RSA

Dobbiamo dimostrare che $M = C^d \bmod n$, avremo bisogno durante la procedura dell'ugualianza:

$$\begin{aligned} e \cdot d &= k \cdot \phi(n) + 1 \\ d &= e^{-1} \bmod \phi(n) \\ e \cdot d &= e \cdot e^{-1} \bmod \phi(n) \\ e \cdot d &= 1 \bmod \phi(n) \\ e \cdot d &= k \cdot \phi(n) + 1 \end{aligned}$$

Infatti ricordiamo che d è l'inverso moltiplicativo di e modulo $\phi(n)$ e questo vuol dire che il prodotto $e \cdot d - 1$ è un multiplo di $\phi(n)$ quindi possiamo scriverlo come $k \cdot \phi(n)$.

$$\begin{aligned} C^d \bmod n &= (M^e \bmod n)^d \bmod n \\ &= M^{e \cdot d} \bmod n \\ &= M^{k \cdot \phi(n)+1} \bmod n \\ &= M \cdot M^{k \cdot \phi(n)} \bmod n \\ &= M \cdot M^{k \cdot (p-1) \cdot (q-1)} \bmod n \\ &= (M \cdot (M^{(p-1)})^{k \cdot (q-1)}) \bmod n \end{aligned}$$

Possiamo verificare l'ugualianza:

$$M \cdot M^{k \cdot (p-1)(q-1)} \bmod p = M \bmod p$$

Infatti, se $M \bmod p = 0$ allora entrambi i membri sono uguali a *0*, nel caso contrario, ovvero nel caso in cui $M \bmod p \neq 0$ che:

$$\begin{aligned} (M \cdot M^{k \cdot (p-1)(q-1)}) \bmod p &= (M \cdot (M^{(p-1)^{k \cdot (q-1)}})) \bmod p \\ &= (M \cdot (M^{p-1} \bmod p)^{k \cdot (q-1)}) \bmod p \\ &= (M \cdot (1)^{k \cdot (q-1)}) \bmod p \;||\; \Leftarrow \text{per il piccolo teorema di Fermat} \\ &= M \bmod p \end{aligned}$$

Allo stesso modo vale in maniera *equivalente*:

$$M \cdot M^{k\cdot(p-1)(q-1)} \bmod q = M \bmod q$$

Le quantità C^d e M sono congrue sia in $\bmod p$ che in $\bmod q$, in altri termini le quantità C^d e M hanno gli stessi **resti** nella divisione con p e q, ma poiché $gcd(p,q) = 1 \wedge p \cdot q = n$ e utilizzando il ***Chinese remainder theorem*** avremo che $C^d \equiv M \bmod n$ riuscendo a verificare la **reversibilità** di RSA.

Efficenza

A differenza di *DH* e quindi non necessitando che p e q siano generati come *strong prime*, ovvero $p, q = 2 * k + 1$ fa in modo che p e q possano essere determinati velocemente. Andando ad analizzare la creazione della della *chiave pubblica* e della *chiave privata*, queste computazioni vengono eseguite una sola volta. Cifratura e decifratura richiedono calcoli esponenziali modulari, che sono operazioni *efficentemente implementate* ma che coinvolgono numeri non gestibili in aritmentica di macchina. Ci interessa trovare un e adatto ai nostri scopi, ovvero che siamo **coprimo** con $\phi(n)$.

- scegliere un numero $e > max\{p, q\}$, infatti andando a calcolare $\phi(n) = (p-1) \cdot (q-1)$ avremo che i fattori di $\phi(n)$ sono tutti minori di p e q.
- procedere per "tentativi casuali", grazie al fatto che il calcolo del *MCD* è molto rapido.
- fissare a priori un esponente piccolo (3 o 5) e continuare a generare primi p e q fino a quando non si verifica la condizione $gcd(e, (p-1) \cdot (q-1)) = 1$, nelle implementazioni **reali** si procede in questo modo partendo da un e più elevato.

La **Sicurezza** di RSA risiede, dal punto di vista strettamente *matematico*, della difficoltà di **fattorizzare** numeri interi di grandi dimensioni, infatti se ne esistessero *Eve* potrebbe ottenere p e q in modo da otterene prima $\phi(n)$ e successivamente d. Il viceversa non è dimostrato, non si è stati però in grado di violare efficaciemente RSA attraverso la fattorizzazione di $\boldsymbol{n}$. Altra faccenda è invece parlare della **correttezza**

implementativa di RSA.

Andiamo ora ad elencare alcune **vulnerabilità** nel caso in cui RSA sia impletato in maniera errata:

- **Malleabilità**: ricordiamo che un algoritmo di cifratura $\boldsymbol{E}$ è *malleabile* se dato un testo cifrato $C_1 = E(M_1)$ è possibile creare un secondo testo cifrato $C_2 = E(M_2)$ tale per cui ci sia una *corrispondenza "significativa"* tra M_1 e M_2. Il protocollo RSA di base è **malleabile**. In questo caso avremo un attacco del tipo ***Chosen Ciphertext Attack*** infatti il prodotto di due ciphertext corrisponde al prodotto di due plaintext. Per dimostrarlo andiamo a definire un testo cifrato $C_1 = M_1^e \bmod n$, a questo punto creiamo un testo cifrato "intermedio", ad esempio con $M_1' = 2$ avremo che $C_1' = 2^e \bmod n$. Possiamo quindi ottenere $C_2 = C_1 * C_1' = (M_1^e \bmod n) \cdot (M_1'^e \bmod n) = (M_1 \cdot M_1')^e \bmod n$ ovvero un ciphertext arbitrario, nel caso sia present un *Decipher Oracle* potremmo decifrare il nostro C_2 a quel punto abbiamo facilmente modo di ottenere M_1:

$$\begin{aligned} C_2^d \bmod n &= (C_1 \cdot C_1')^d \bmod n \\ &= ((M_1^e \bmod n) \cdot (2^e \bmod n))^d \bmod n \\ &= ((M_1^e)^d \cdot (2^e)^d) \bmod n \\ &= (M_1 \cdot 2) \bmod n \end{aligned}$$

  È quindi possibile moltiplicare per l'inverso $M_1' \bmod n$ per ottenere l'M_1 desiderato. In letteratura questo tipo di attacco è anche nota come proprietà ***Omomorfica***.

- **Caso di *e* troppo piccolo**: nel momento in cui e è troppo piccolo e contemporaneamente il testo da cifrare è anch'esso molto breve, in quest'ottica se $e = 3$ e il messaggio è breve può succedere che $M^e < n$, in questo caso succede che il modulo n non riduca il risulta di $M^e \bmod n$ e che quindi avremo che $C = M^e$. In questo caso *Eve* può calcolare la radice cubica di C. Per questa ragione, lo standard ***FIPS*** prevede che e non sia minore di $2^{16} + 1 = 65537$. Questa problematica unita a quella precedente possono portare a tipologie di attacco del tipo: ***Hastad's Broadcast Attack*** (bisogna combinare le due vulnerabilità tramite il *CRT*).

- **Possibilità di Fattorizzare n**: nel caso in cui p e q vengano scelti con caratteristiche non idonee è possibile modellarli in modo tale da ottenere la fattorizzazione di n. Identifichiamo come **malconfigurazione** il fatto che p e q vengano scelti **"molto vicini"**. Andiamo ad analizzare il contesto:
  $n = p * q \quad \Rightarrow p \neq q$ possiamo dire che: $p > q$

$$\begin{aligned} n &= (\frac{p+q}{2})^2 - (\frac{p-q}{2})^2 \\ &= \frac{p^2 + q^2 + 2pq - p^2 - q^2 + 2pq}{4} \\ &= \frac{4 * pq}{4} = p \cdot q \end{aligned}$$

  È quindi possibili visualizzare n come differenza di quadrati del tipo $n = x^2 - y^2$, per poter utilizzare questa nozione per portare un attacco su RSA dobbiamo cercare di capire con quanta facilità è possibile calcolare la quantità: $\frac{p+q}{2}$ o $\frac{p-q}{2}$ senza ovviamente conoscere p e q. La possibilità più immediata è quella di tentare un *brute-force* che permettere iterando su una delle due (ad esempio x^2) di ottenere l'altra. Calando ad ogni iterazione $y^2 = x^2 - n$. Andando nel dettaglio:

  1. assegnamo a x_0 un valore arbitrario x'. Il valore iniziale di x deve essere sensato per il caso d'uso, consideriamo quindi:

$$\begin{aligned} \tfrac{p+q}{2} &> \sqrt{n} \\ \Rightarrow\ x' &= \lceil \sqrt{n} \rceil \end{aligned}$$

  2. calcoliamo la quantità $z_i = x_i^2 - n$. Nel caso in cui z_i è un quadrato perfetto in $\mathbb{Z}_n$ allora passiamo al punto *4*.
  3. Poniamo $x_i = x_{i-1} + 1$ e torniamo al passo precedente.
  4. possiamo calcolare e restituire i due fattori: $x + \lceil \sqrt{z_i} \rceil$ e $x - \lceil \sqrt{z_i} \rceil$.

  Analizzando questo attacco a *forza bruta* è interessante andare a calcolare sotto quali condizioni l'algoritmo riesca a **fattorizzare** in "tempi ragionevoli". Ovvero che la differenza tra p e q risulti $\mathcal{O}(\sqrt[4]{n})$, in termini di cifre significa che p e q coincidano nelle metà più significative.

- **Vulnerabilità Dipendenti dai Generatori Casuali**: nel 2012 è stato fatto uno studio su 4.7 milioni di moduli RSA di 1024 bit ed è risultato che i dublicati non sono infrequenti e in alcuni casi (12720) si è notato che i due moduli avevano un fattore in comune, questo permetterebbe la fattorizzazione di ben due n e quindi l'inutilià della chiave pubblica generata in quanto chiunque riuscirebbe a leggere il ciphertext. Infatti è sufficiente calcolare $gcd(n_1, n_2) = f$, dove f è il fattore primo in comune per riuscire ad ottenere prima $\phi(n)$ e successivamente d. La causa di questo attacco è da imputare per la maggiore alle **debolezze dei generatori (pseudo)casuali** coinvolti nella scelta dei numeri primi

È quindi importante, per non incorrere in vulnerabilità di applicare **aspetti implementativi** apparentemente marginali, ma che bisogna tenere in considerazione quando si va a sviluppare un architettura che si basa su **RSA**. Quali:

- la **generazione** di **numeri casuali** deve essere effettuata dopo che si ha accumulato abbastanza **entropia** da garantire che le eventuali collisioni non abbiano probabilità maggiore di quanto previsto teoricamente.
- i fattori p e q non devono essere troppo vicini per evitare che venghino fattorizzati tramite l'**algoritmo di fattorizzazione di Fermat**.
- gli esponenti e e d non devono essere piccoli, per evitare di rendere inefficace la riduzione in modulo n.
- la malleabilità del protocollo deve essere eliminata con opportuni accorgimenti.

Miglioramente dell'efficienza di RSA

È possibile effettuare in due modi diversi un'ottimizzazione.

1. il primo lavora nel momento della **cifrazione** ovvero durante il calcolo $C = M^e \bmod n$, infatti sappiamo che l'esponenziale modulare non è altro che uno shift di m posizioni in base alle posizioni dei bit *non 0*, secondo il **FIPS** viene raccomandato che e non sia minore di **65537**, in molti casi reali la scelta ricade

proprio su questo numero in quanto può essere rappresentato come $(2^{16}+1)_{(10)} = 10000000000000001_{(2)}$ e proprio la presenza di molte cifre a 0 rende **più efficiente il calcolo dell'esponenziale modulare**.

2. il secondo metodo riguarda invece la fase di **decifrazione** infatti è previsto che *Alice* ricavi il plaintext tramite $M = C^d \bmod n$ possiamo però utilizzare il ***CRT*** per ridurre la dimensione degli esponenti.

$$M_p = C^d \bmod p \quad \text{e} \quad M_q = C^d \bmod q$$

e possiamo risalire al valore di M tramite:

$$M = ((q \cdot (q^{-1} \bmod p)) \cdot M_p + (p \cdot (p^{-1} \bmod q)) \cdot M_q) \bmod n$$

in cui i coefficenti $q \cdot (q^{-1} \bmod p)$ e $p \cdot (p^{-1} \bmod q)$ possono essere precalcolati.

È possibile ottimizzare in maniera maggiore precalcolando $s = d \bmod (p-1)$ e $t = d \bmod (q-1)$. Possiamo analizzare che $d = s + a \cdot (p-1)$ e che dunque possiamo semplificare i calcoli:

$$\begin{aligned} C^d \bmod p &= C^{s+a\cdot(p-1)} \bmod p \\ &= C^s \cdot C^{(p-1)^a} \bmod p \\ &= C^s \cdot 1^a \bmod p \\ &= C^s \bmod p = M_p \end{aligned}$$

Possiamo svolgere gli stessi calcoli, analogamente, per t andando a sostituire il modulo p con il valore $\boldsymbol{q}$ andando ad ottenere: $M_q = C^t \bmod q$. E quindi calcolando M_p ed M_q:

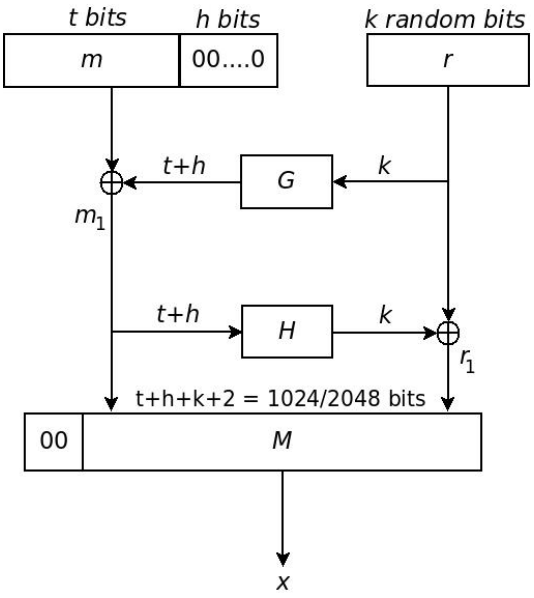
$$M_p = C^s \bmod p \quad \text{e} \quad M_q = C^t \bmod q$$

Questa computazione permette di calcolare gli esponenziali su numeri di grandezza dimezzata.

Optimal Asymmetric Encryption Padding - OAEP

Permette di aggiungere alla cifratura di RSA un informazione addizionale chiamata **padding**, costituita da sequenze *(pseudo)casuali* ed è quindi sempre più determinante

avere dei generatori affidabili.



Consideriamo n è il numero di bit nel modulo RSA, h e k sono costanti intere fissate dal protocollo ed m è il messaggio in chiaro, ovvero una stringa lunga $n - k - h - 2$. Definiamo ora $\boldsymbol{G}$ e $\boldsymbol{H}$ come due funzioni di *hash crittografico*. Andiamo ad analizzare i passaggi per cifrare:

1. al messaggio m va applicato un *padding* di h zeri per arrivare alla lunghezza di $n - k - 2$
2. definiamo r come una stringa casualmente generata di k bit
3. r verrà poi espansa da k a $n - k - 2$ bit tramite la funzione di hash $\boldsymbol{G}$
4. $m_1 = m \oplus G(r)$
5. tramite $\boldsymbol{H}$ m_1 viene ridotta da $n - k - 2$ bit a k bit.
6. $r_1 = r \oplus H(m_1)$
7. il messaggio finale, che verrà successivamente convertito in un numero x da mandare come input ad RSA è "$00 \mid\mid m_1 \mid\mid r_1$"

La **decifrazione** è definita dalle proprietà dello ***xor*** logico.

4.4 Sistema Crittografico Asimmetrico di Rabin

Poco dopo l'introduzione di *RSA*, **Michael O. Rabin** ha sviluppato un algoritmo crittografico a cui appartiene la proprietà teorica di essere equivalente alla fattorizzazione, ovvero un algoritmo *polynomial time* in grado di violare il sistema di *Rabin* potrebbe essere utilizzato per la fattorizzazione di interi (e viceversa), ma, come contro, presenta una limitazione in quanto la decifrazione di un ciphertext arbitrario produce **quattro possibili** candidati come **plaintext**. Andiamo ad analizzare le quattro possibili radici di un numero $y \in \mathbb{Z}_p^*$ attraverso un esempio numerico.

Definiamo: $p = 13 \;\; q = 23 \;\; n = p \cdot q = 299$

Prendiamo un elemento $x \in \mathbb{Z}_p^*$: $x = 27$

Calcoliamo il suo quadrato: $y = x^2 \bmod n = 131$

Sappiamo che in $\mathbb{R} \Rightarrow \sqrt{131} = \pm x$

Allora anche in $\mathbb{Z}_n^*$ saranno presenti due radici: $(x, -x)$

Calcoliamo $-x = n - x = 272 \Rightarrow (-x)^2 \bmod n = 131$

Precalcoliamo $c_p = q \cdot (q^{-1} \bmod p) = 92 \quad c_q = p \cdot (p^{-1} \bmod q) = 208$

Consideriamo ora che $n = p \cdot q$ utilizzando il *CRT* sappiamo che data l'equazione $\exists\, C \mid C \equiv y \bmod n,\; C = c_p * a_1 + c_q * a_2$, prima di tutto andiamo a calcolarci le quantità a_1 e a_2, come $a_i = y \bmod n_i, \quad i = \{p, q\}$, a_1 è il corrispondente di y in $\mathbb{Z}_p^*$ e, ovviamente a_2 è il corrispondente di y in $\mathbb{Z}_q^*$. Allora sia per a_1 che per a_2 saranno presenti 2 radici, rispettivamente in $\mathbb{Z}_p^*$ e in $\mathbb{Z}_q^*$, possiamo calcolarli come:

$$x_{p^1} = y^{(2^{-1} \bmod p)} \bmod p = 1$$
$$-x_{p^1} = x_{p^2} = p - x_{p^1} = 12$$
$$x_{q^1} = y^{(2^{-1} \bmod q)} \bmod q = 4$$
$$-x_{q^1} = x_{q^2} = q - x_{q^1} = 19$$

Per trovare le radici di y in $\mathbb{Z}_n^*$ applicando il ***CRT*** avremo:

$$x_1 = c_p * x_{p^1} + c_q * x_{q^1} = 27$$
$$x_2 = c_p * x_{p^1} + c_q * x_{q^2} = 157$$
$$x_3 = c_p * x_{p^2} + c_q * x_{q^1} = 142$$
$$x_4 = c_p * x_{p^2} + c_q * x_{q^2} = 272$$

Alla fine avremo che $y = x_i^2 \bmod n = 131 \ \ \forall i = 1, 2, 3, 4$, siamo riusciti a "dimostrare" che preso un *quadrato perfetto* in $\mathbb{Z}_n*$ con n che è il prodotto di due numeri *coprimi* tra loro, il quadrato perfetto avrà quattro **radici** in $\mathbb{Z}_n*$, due delle quali saranno fra loro **congrue modulo p** e due **congure modulo q**. Riuscendole ad ottenere applicando il *CRT*.

Descriviamo ora l'**algorimo generale**, anche in questo caso verrà diviso in generazione delle chiavi, che necessita come input un solo parametro, ovvero la lunghezza in bit di n. E una parte di scambio di informazioni che quindi necessiterà di utilizzare cifratura e decifratura.

- **Generazione delle chiavi: *Alice***
  1. genera due numeri primi a caso di lunghezza $\frac{n}{2}$ bit tale che $p \equiv 3 \bmod 4$ e $q \equiv 3 \bmod 4$.
  2. calcolo di $n = p \cdot q$.
  3. diffonde n come **chiave pubblica** e conserva la coppia *(p, q)* come **chiave segreta**.
- **Cifratura di un messaggio: *Bob***
  1. si procura la chiave pubblica n di *Alice*.
  2. calcola $C = M^2 \bmod n$.
  3. invia C ad *Alice* come messaggio cifrato.
- **Decifrazione del messaggio *C*: *Alice***
  1. *Alice* calcola $M_p = C^{\frac{p+1}{4}} \bmod p$ e $M_q = C^{\frac{q+1}{4}} \bmod q$, notiamo che M_p è una delle *radici quadrate di C* modulo p perché:

$$\begin{aligned} M_p^2 \bmod p &= (C^{\frac{p+1}{4}})^2 \bmod p \\ &= C^{\frac{p+1}{2}} \bmod p \\ &= C^{\frac{p-1}{2}} \cdot C \bmod p \\ &= ((M^2 \bmod n)^{\frac{p-1}{2}} \cdot C) \bmod p \\ &= (((M^2 \bmod n) \bmod p)^{\frac{p-1}{2}} \cdot C) \bmod p \\ &= ((M^2 \bmod p)^{\frac{p-1}{2}} \cdot C) \bmod p \\ &= (M^{p-1} \cdot C) \bmod p \\ &= (1 \cdot C) \bmod p \\ &= C \bmod p \end{aligned}$$

Analogamente M_p è una delle *radici quadrate di* C modulo q.

2. riprendendo il discorso del *CRT* possiamo calcolare le quattro radici di C modulo n e una di questi valori sarà il messaggio originale.

$$M_1 = ((q \cdot (q^{-1} \bmod p)) * M_p) + (p \cdot (p^{-1} \bmod q) * M_q) \bmod n = (c_p \cdot M_p + c_q \cdot M_q) \bmod n$$
$$M_2 = n - M_1 = ((-c_p) \cdot M_p + (-c_q) \cdot M_q) \bmod n$$
$$M_3 = ((q \cdot (q^{-1} \bmod p)) * M_p) - (p \cdot (p^{-1} \bmod q) * M_q) \bmod n = (c_p \cdot M_p + (-c_q) \cdot M_q)$$
$$M_4 = n - M_3 = ((-c_p) \cdot M_p + c_q \cdot M_q) \bmod n$$

Utilizzando il *CRT* possiamo verificare che, effettivamente, M è effettivamente una delle quattro radici di C modulo n.

Equivalenza con il problema della fattorizzazione

Dobbiamo dimostrare che se siamo in grado di decifrare (con un incertezza 1 su 4) allora possiamo fattorizzare n, il viceversa è ovvio. Abbiamo visto che le quattro radici di C mod n corrispondono alle quattro possibili combinazioni delle due radici modulo p e delle due radici modulo q:

1. M_1 corrisponde alla coppia $(M_p, M_q) = (C^{\frac{p+1}{4}} \bmod p, C^{\frac{q+1}{4}} \bmod q)$.

2. M_2 corrisponde alla coppia $(p - M_p, q - M_q)$.

3. M_3 corrisponde alla coppia $(M_p, q - M_q)$.

4. M_4 corrisponde alla coppia $(p - M_p, M_q)$.

L'obiettivo è fattorizzare un numero n dato in input e considerare di possedere una scatola **black-box** che contenga l'algoritmo di dato un numero C, resido quadratico in modulo n, restituirti una delle quattro radici.
La riduzione è un algoritmo tipo **Las Vegas** e funziona in questo modo:

1. genero a caso un numero $r \in \mathbb{Z}_n, r \neq 0$.

2. se $m = gcd(r, n) \neq 1$ restituisco m e n/m.

3. altrimenti considero r come un "messaggio", calcolo $C = r^2 \bmod n$ e sottopongo C alla black box.

4. se r' è il valore restituito dalla black-box, calcolo $m = gcd(r - r', n)$.

5. se $m > 1$ e $m \neq n$ restituisco i fattori m e n/m, altrimenti ritorno al passo *1.*

La correttezza è che la *black-box* ritorna r' che è una delle quattro radici di $C = r^2 \bmod n$ e che può essere uno dei quattro modi che abbiamo per combinare le due radici di $r^2 \bmod p$ e le due radici di $r^2 \bmod q$. Consideriamo che $r = (r_p, r_q) = (M_p, M_q)$ (esempio senza perdere di generalità).

1. $r' = (r_p, r_q) = r$ allora avremo che $r - r' = 0$ ovvero che seguendo il CRT avremo che $(r - r') \bmod p = r_p - (p - r_p) \bmod p = 2r_p \bmod p$ e che $(r - r') \bmod q = r_q - (q - r_q) \bmod q = 2r_q \bmod q$ e che quindi $n = m$ e che quindi non riusciamo a ad ottenere una risposta dall'algoritmo.

2. $r' = (-r_p, -r_q) = -r$ allora avremo che $r - r' = 2r$ e quindi otteniamo $gcd(r, n) = 1$ e anche qui non riusciamo a determinare i fattori di n.

3. $r' = (r_p, -r_q)$ allora avremo che $r - r' = (r_p, r_q) - (r_p, -r_q) = (r_p - r_p, r_q - (-r_q)) = (0, 2r_q)$ ma avere il fatto in modulo p a *0* implica che nel CRT il suo fattore modulo p sia $r_p = C^{\frac{p+1}{4}} \bmod p = 0$ che implica che $C^{\frac{p+1}{4}}$ sia multiplo con p e quindi avremo che $gcd(r', n) = p$.

4. $r = (-r_p, r_q)$ in questo caso avremo l'esatto opposto del punto precedente, ovvero che $r - r' = (2r_p, 0)$ in questo caso avremo che quindi nel fattore $r_p = C^{\frac{q+1}{4}} \bmod q = 0$ del CRT avremo che $C^{\frac{q+1}{4}}$ sia multiplo con q e quindi avremo che $gcd(r', n) = q$.

Riduzioni Polinomiali

Per riuscire a dimostrare che un algoritmo è "uguale" ad un altro, in questo caso dobbiamo dimostrare che l'**algoritmo di fattorizzazione** è "uguale" all'**algoritmo per la risoluzione di Rabin**, bisogna dimostrare che l'uno e l'altro abbiano uguale **complessità**. Nel caso di dimostrare che saper risolvere **Rabin** equivale a saper **FATTORIZZARE** è immediato infatti se sappiamo fattorizzare un numero n allora otteniamo p e q e quindi abbiamo la chiave privata di *Alice*. Per dimostrare, invece, che risolvere **Rabin** vuol dire risolvere il problema della **FATTORIZZAZIONE** abbiamo utilizzato il sistema della **black box**, ovviamente affinché questo funzioni la *black box* deve ritornare una della quattro radici in maniera casuale.

Capitolo 5

Firma Digitale

La *firma digitale* è il metodo di **autenticazione** basato su crittografia a chiave pubblica. In generale si tratta di una **"cifratura"** con la **chiave privata**. L'**autenticità del mittente** deriva dal fatto che solo con la corrispondente **chiave pubblica del mittente** si possa decifrare un messaggio "cifrato" M.

5.1 Firma Digitale con RSA

L'osservazione fondamentale è il ruolo perfettamente **simmetrico** degli esponenti e e d, infatti essi sono l'uno l'inverso dell'altro in $\mathbb{Z}_n^*$. È quindi possibile ***firmare*** un messaggio M con la propria **chiave privata**, in questo caso **non** avremo **riservatezza** sul messaggio, infatti essendo che il messaggio M si potrà decifrare con la chiave pubblica, quindi nota a chiunque, essa servirà unicamente per validare che quel messaggio è stato inviato dal proprietario della corrispondente chiave privata.

$$F = M^d \bmod n \qquad M = F^e \bmod n$$

Naturalmente anche se un *destinatario* decifrasse il messaggio F con la chiave pubblica del *mittente* non potrebbe validare il messaggio è quindi necessario associare alla quantità F il messaggio originale M per poter poi effettuare il confronto tra i due, ecco quindi derivata la "definizione" di ***digital signature***: $DS = (M, F)$.
La garanzia è che senza una copia di d è computazionalmente intrattabile **forgiare**

una coppia $(\overline{M}, \overline{F})$ dove $\overline{F} = \overline{M}^d \bmod n$.

Protocollo di Base

- **Firma del Messaggio: *Alice***
  1. calcolo $F = M^d \bmod n$.
  2. invio della coppia (M, F).
- **Verifica della Firma: *Bob***
  1. ottiene la chiave pubblica di *Alice* (n, e).
  2. calcola $M' = F^e \bmod n$.
  3. accetta se e solo se $M = M'$.

In un implementazione "reale" la coppia non contiene la firma del messaggio M, ma contiene la **firma** di un **fingerprint** di M, che viene normalmente ottenuto mediante l'applicazione di una **funzione di hash crittografica**. Viene normalmente effettuato per **ridurre** sensibilmente la **dimensione** dei numeri senza però **perdere sicurezza** a patto che la funzione di **hash** sia **resistente alle collisioni**.
Ovvero la funzione deve essere ***Second Pre-Image Resistant*** quindi che dato un M deve essere computazionalmente proibitivo trovare un $\overline{M} \mid H(M) = H(\overline{M})$.

Protocollo di Base con Hash

- **Firma del Messaggio: *Alice***
  1. calcola $H = hash(M)$
  2. calcolo $F = H^d \bmod n$.
  3. invio della coppia (M, F).
- **Verifica della Firma: *Bob***
  1. ottiene la chiave pubblica di *Alice* (n, e).
  2. calcola $H' = F^e \bmod n$.

3. calcola $H = hash(M)$
4. accetta se e solo se $H = H'$.

Blinding Attack: ***Eve*** Cerchiamo un numero R tale che $\overline{M} = R^e \cdot M \bmod n$, *Eve* "convince" (tramite attacchi di *social engineering*) *Alice* a firmarlo ottenendo $F = \overline{M}^d \bmod n$. Poiché:

$$\begin{aligned} F &= \overline{M}^d \bmod n \\ &= (R^e \cdot M \bmod n)^d \bmod n \\ &= R^{ed} \cdot M^d \bmod n \\ &= R^{1+\phi(n)} \cdot M^d \bmod n \\ &= R \cdot R^{\phi(n)} \cdot M^d \bmod n \;||\; R^{\phi(n)} \bmod n = 1 \text{ teorema di } \textbf{Eulero} \\ &= R \cdot 1 \cdot M^d \bmod n \\ &= R \cdot M^d \bmod n \end{aligned}$$

A *Eve* è sufficiente calcolare $F \cdot R^{-1} \bmod n$ per ottenere proprio $M^d \bmod n$, la parte difficoltosa è riuscire a nascondere in un messaggio apparentemente innocuo un messaggio pericoloso. La "debolezza" di questo attacco risiede nell'utilizzo di un ***hash*** del messaggio per effettuare la firma. Non sono tutt'ora noti attacchi funzionanti al protocollo con hash. La possibilità di effettuare attacchi si riduce ancora nel momento in cui al messaggio viene aggiunto del *padding "casuale"*. L'idea è sviluppata all'interno dello standard ***Probabilistic Signature Scheme - PSS*** nel quale, se viene applicata una funzione *hash* il messaggio da firmare avrà una lunghezza molto minore della lunghezza massima consentita da *RSA*, questo permette di riempire lo "spazio" rimanente con **bit casuali**. Un esempio è ***RSA-OAEP***.

5.2 Firma Digitale con ElGamal

Definiamo la chiave **pubblica** e **privata**:

- **Chiave Privata**: a
- **Chiave Pubblica**: (p, g, A) dove p è un primo, g è un generatore di un sottogruppo di $\mathbb{Z}_p^*$ e $A = g^a \bmod p$ dove a è la *chiave privata* ed è un numero tale che $a \in \{2, 3, ..., p-2\}$

Protocollo di Base

- **Firma del Messaggio: *Alice***
  1. seglie un numero k a caso tale che $gcd(k, p-1) = 1$.
  2. calcola le quantità $r = g^k \bmod p$ e $s = k^{-1} \cdot (M - a \cdot r) \bmod (p-1)$.
  3. il messaggio firmato è $(M, (r, s))$.
- **Verifica della Firma: *Bob***
  1. si procura la chiave pubblica di *Alice*: (p, g, A).
  2. calcola le due quantità $x_1 = A^r \cdot r^s \bmod p$ e $x_2 = g^M \bmod p$.
  3. accetta la firma come autenticata se e solo se $x_1 = x_2$.

La **Correttezza** del metodo è data da:

$$\begin{aligned}
r^s \bmod p &= (g^k \bmod p)^s \bmod p \\
&= (g^k \bmod p)^{(k^{-1}\cdot(M-ar)) \bmod (p-1)} \bmod p \\
&= (g^k \bmod p)^{(k^{-1}\cdot(M-ar)-(p-1)h^*)} \bmod p \\
&= (g^k)^{(k^{-1}\cdot(M-ar))} \cdot (g^k)^{-(p-1)\cdot h} \bmod p \\
&= g^{M-ar} \cdot (g^{-kh})^{p-1} \bmod p \\
&= g^M \cdot g^{-ar} \bmod p \\
&= g^M \cdot (g^a \bmod p)^{-r} \bmod p \\
&= g^M \cdot A^{-r} \bmod p
\end{aligned}$$

dove h^* è quoziente della divisione di $k^{-1} \cdot (M - ar)$ e per $p-1$ seguendo la combinazione lineare $R = x - \lfloor \frac{x}{y} \rfloor * Q$.

$$\begin{aligned} x_1 &= A^r \cdot r^s \bmod p \\ &= A^r \cdot (g^M \cdot A^{-r} \bmod p) \bmod p \\ &= g^M \bmod p \\ &= x^2 \end{aligned}$$

Configurazioni di Sicurezza

- **Uso singolo di K**: il valore segreto k deve essere usato una sola volta. Se M_1 e M_2 vengono firmati con lo stesso k ad *Eve* basta impostare il sistema di equazioni:

$$\begin{cases} a \cdot r_1 + k \cdot s_1 = M_1 \pmod p \\ a \cdot r_2 + k \cdot s_2 = M_2 \pmod p \end{cases}$$

  In questo modo l'unica soluzione al sistema non è altro che la coppia (a, k) ovvero il k utilizzato per la generazione di r e s e la chiave private a.

- **Valore hash del messaggio**: il protocollo deve essere usato con un **hash del messaggio**, in caso contrario *Eve* può eseguire un ***Message Forgery Attack***, ovvero produrre un messaggio correttamente firmato senza conoscere la chiave privata di *Alice*. Si possono infatti identificare una coppia (r, s) tale per cui il messaggio M creato abbia come firma proprio la coppia (r, s).

  1. *Eve* sceglie due numeri x e y con $gcd(y, p-1) = 1$
  2. sceglie un r tale per cui valga:

$$\begin{aligned} r &= g^x \cdot A^y \bmod p \\ &= g^x \cdot (g^a \bmod p)^y \bmod p \\ &= g^{x+a \cdot y} \bmod p \end{aligned}$$

  3. una volta ottenuto r possiamo calcolare $s = -r \cdot y^{-1} \bmod (p-1)$.
  4. *Eve* infatti sa che *Bob* eseguirà il seguente controllo:

$$A^r \cdot r^s \bmod p \stackrel{?}{=} g^M \bmod p$$

e dunque pone $M = x \cdot s \bmod (p-1)$.

5. in questo modo risulta:

$$\begin{aligned} A^r \cdot r^s \bmod p &= (g^a \bmod p)^r \cdot (g^{x+a\cdot y})^s \bmod p \\ &= g^{a\cdot r} \cdot g^{(x+a\cdot y)\cdot s} \bmod p \\ &= g^{a\cdot r+x\cdot s+a\cdot y\cdot s} \bmod p \\ &= g^{a\cdot r+x\cdot s} \cdot g^{a\cdot y\cdot(-r\cdot y^{-1}-h'\cdot(p-1))} \bmod p \\ &= g^{a\cdot r+x\cdot s} \cdot g^{a\cdot y\cdot -r\cdot y^{-1}} \cdot g^{h(p-1)} \bmod p \\ &= g^{a\cdot r+x\cdot s} \cdot g^{-r\cdot a} \bmod p \\ &= g^{a\cdot r} \cdot g^{x\cdot s} \cdot g^{a\cdot -r} \bmod p \\ &= g^{x\cdot s} \bmod p \\ &= g^{x\cdot -(r\cdot y^{-1}+h'\cdot(p-1))} \\ &= g^{x\cdot -r\cdot y^{-1}} \cdot g^{h'(p-1)} \bmod p \\ &= g^M \bmod p \end{aligned}$$

Abbiato riutilizzato la definizione di modulo: $R = x - \lfloor \frac{x}{y} \rfloor * Q$ andando a sostituire $s = -r \cdot y^{-1} \bmod (p-1)$ con $s = -r \cdot y^{-1} - h \cdot (p-1) = -(r \cdot y^{-1} + h \cdot (p-1))$ e grazie al *piccolo Teorema di Fermat* abbiamo semplificato con $s = -r \cdot y^{-1}$.

Per riuscire a difendersi da questo tipo di attacco possiamo utilizzare una funzione di hash, per ottenere un H per poi firmarlo. Più precisamente, dopo aver scelto k e calcolato r, *Alice* firma $H(M||r)$ ovvero il messaggio M concatenato al valore r, in questo modo *Eve* non potrebbe porre $M = x \cdot s \bmod (p-1)$, perché non è questo che *Bob* usa nella verifica. *Eve* dovrebbe trovare un messaggio M tale che $H(M||r) = x\cdot s \bmod (p-1)$ ma questo implicherebbe che ***H* non** sia ***First Pre-Image Resistant***. In oltre bisogna che *Bob* controlli il valore di r se no, *Eve* potrebbe produrre firme apparentemente autentiche su qualsiasi messaggio, nel momento in cui disponga della firma autenticata (r, s) di un solo messaggio M.

5.3 Firma Digitale con Digital Signature Algorithm

Il ***Digital Signature Algorithm*** anche detto ***DSA*** è un metodo definito con precisione dal ***National Institute of Standard and Technology*** (***NIST***) e adottato come standard dal 1994. A differenza degli altri protocolli **DSA** prevede fin da subito l'utilizzo di una funzione di **hash crittografico**, nello specifico *SHA-1*

Protocollo Originale

- **Generazione delle Chiavi: *Alice***
  1. genera un numero q di 160 bit.
  2. genera un numero p di 1024 bit tale che $gcd(q, p-1) > 1$.
  3. determina un elemento di $g \in \mathbb{Z}_p^*$ di ordine q ovvero un generatore di $\mathbb{Z}_q \subseteq \mathbb{Z}_p^* \approx 2^{160}$ elementi.
  4. scegli a caso un $a \in \mathbb{Z}_q$
  5. calcola $A = g^a \bmod p$
  6. pubblica (p, q, g, A) e tieni riservato il numero a.
- **Firma del Messaggio: *Alice***
  1. dato il messaggio M, calcola $m = SHA1(M)$
  2. sceglie un $k \in \mathbb{Z}_p^*$ uniformemente a caso.
  3. calcola $r = (g^k \bmod p) \bmod q$ e $s = k^{-1} \cdot (m + a \cdot r) \bmod q$
  4. se anche uno solo dei due valori (r o s) è *0*, scegli un valore diverso di k (punto 2).
  5. altrimenti invia il messaggio $(m, (r, s))$.
- **Verifica della Firma: *Bob***
  1. si procura la chiave pubblica di *Alice* (p, q, g, A).
  2. controlli che risulti $0 < r,\ s < q$.
  3. calcola $x = SHA1(M) \cdot (s^{-1} \bmod p)$

4. calcola $y = r \cdot (s^{-1} \bmod p)$.
5. esegue il controllo $(g^x \cdot A^y \bmod p) \bmod q \stackrel{?}{=} r$ e se verificato accetta la firma come autentica.

La **sicurezza** di **DSA** dipende dalla difficoltà del logaritmo discreto su sottogruppi di q elementi. Il protocollo originale, in qui q veniva scelto nell'intervallo $(2^{159}, 2^{160})$ fornisce quindi $\log_{\sqrt{2^{160}}} = 80$ bit di **sicurezza**. Le **specifiche del protocollo attuale** modificano solo la lunghezza di p e q $p \in \{1024, 2048, 3072\}$ mentre $q \in \{160, 224, 256\}$ come funzione di **hash crittografico** può essere utilizzata qualunque funzione specificata nelle pubblicazioni ***FIPS 180***. Alcune curiosità è che se k è stata compromessa, anche la chiave privata di *Alice* è stata compromessa, mentre se r o s è uguale a *0* allora la chiave privata viene compromessa.

5.4 Autenticità delle chiavi pubbliche

Quando *Bob* si procura la chiave pubblica di *Alice* deve essere certo che la chiave sia **effettivamente** di *Alice*. Ci sono due approcci per ottenere questo risultato il primo su cui si basa ***TLS/OpenSSL***, ovvero che necessita di infrastrutture chiamate ***Certificate Authority - CA***, mentre l'altro che viene utilizzato da ***GnuPG/OpenPGP***.

5.4.1 Approccio TLS

Il *client* contatta un *server* per iniziare una comunicazione, il *server* risponde con un messaggio dove è incluso un **certificato di autenticità** ovvero la *chiave pubblica* del *sever* **firmata** da un **autorità** che funge da **garante** nella comunicazione. A quel punto il *client* controlla che il certificato del *server* sia già in suo possesso e sia *verificato* e *valido*. Se si allora il *client* possiede già la chiave pubblica verificata e l'interazione procede con il *protocollo concordato*. Se no, invece, ma possiede la chiave pubblica del *garante* allora il *client* acquisisce come autentico il certificato memorizzandolo nel cosidetto ***keyring***, e successivamente l'interazione procede con il *protocollo concordato*. Se non si verifica neanche questa condizione allora il *client* deve autenticare il

garante attraverso un **super-garante**. È chiaro che il problema assume cioè connotati **ricorsivi**. Sono presenti però delle ***CA Top Level*** di cui ci si fida senza cercare un loro garante andando quindi ad interrompere l'iterazione. Normalmente queste *chiavi pubbliche* di massimo livello sono salvate nel ***keyring*** quando il ***OS*** viene installato.

5.4.2 GnuPG

In questo caso l'organizzazione non è **verticale** come in nel caso del **TLS**, ma più **orizzontale** (potremmo dire *peer-to-peer*). Si tratta di un'opzione molto più leggera che non potrebbe essere utilizzata negli scopi dove si applica il *TLS*, e viene anche definito il ***web of trust***. Infatti ad ogni chiave pubblica viene sia associato un ***fingerprint***, importando questo nel proprio ***keyring*** aumentano il **grado** di confidenzialità della chiave pubblica. Quindi i "garanti" sono tutti quegli utenti che hanno già importato il ***fingerprint*** di una chiave pubblica all'interno del proprio ***keyring***.

Capitolo 6

Curve Elittiche

Le **curve elittiche** sono oggetto di studio ben prima che fosse scoperto il loro utilizzo in ambito crittografico, infatti questo venne intuito solo negli anni '80. Questo è grazie al fatto che i punti definiti su una **curve elittica** (definita su un *campo finito*) formano un **gruppo** che può essere utilizzato per scopi crittografici.
A scopo didattico partiremo da analizzare le **EC** su $\mathbb{R}$ in modo da poter utilizzare la geometria di esse per "intuire" il significato di certe operazioni. Su campi finiti invece si perde la geometria definita in $\mathbb{R}$ ma rimane l'algebra che è indipendente dal campo sottostante.

Un **campo** (**field**) è un insieme F su cui sono definite due operazioni: l'addizione e la moltiplicazione **logica** che indicheremo con: $\{+, \times\}$. Per esplicitare il campo con le sue operazioni si usa la notazione $(F, +, \times)$, tuttavia quando le operazioni sono ben chiare nel contesto lo indicheremo unicamente con il gruppo. Affinche F sia un **campo** bisogna che sia un **gruppo** rispetto ad entrambe le operazioni contemporaneamente e, in aggiunta, deve anche valere la proprietà **distributiva** della somma rispetto al prodotto. Naturalmente, per l'*elemento neutro dell'addizione* non è richiesto un inverso moltiplicativo. Razionali, reali e complessi sono ben noti campi **infiniti**.
Nell'informatica invece trovano applicazione i **campi finiti**, detti anche ***Galois Field*** (***GF***). Il più semplice al quale si può pensare è ***GF(2)***, ovvero l'insime $\{0, 1\}$ sul quale sono definite le operazioni $\{\oplus, \wedge\}$.

6.1 Campi Finiti

Come abbiamo anticipato il **campo finito** più semplice è il ***GF(2)***. Se definiamo p come un numero primo, l'insieme $\mathbb{Z}_p$ con le operazioni modulari: $\{+_p, \times_p\}$ definete su di esso formano un campo:

$$GF(p) = (\mathbb{Z}_p, +_p, \times_p)$$

Esistono anche **campi finiti** di n elementi se e solo se $n = p^k, \quad k \geq 1$, che verrà indicato con $\mathbf{GF(}p^k\mathbf{)}$, ovviamente nel caso in cui $k = 1$ avremo l'ugualianza:

$$GF(p^k) = \mathbb{Z}_p \iff k = 1$$

Se, invece, $k > 1$ il campo è costituito dai **polinomi** di grado **minore di *k*** con coefficiente in $\mathbb{Z}_p$, le operazioni su questi **campi** sono effettuate **modulo** un **polinomio di grado *k* irriducibile** su $\mathbb{Z}_p$.
Consideriamo come esempio $GF(3^2)$ avremo che quindi i suoi elementi saranno composti dai polinomi di grado al massimo *1* con i coefficienti che saranno modulo *3*:

$$GF(3^2) = \{1, 2, 3, x, x+1, x+2, 2x, 2x+1, 2x+2\} \Rightarrow x^2 + 1$$

In questo caso possiamo utilizzare come **polinomio irriducibile** $x^2 + 1$ e quindi dovremo prendere il resto della divisione intera per il *polinomio irriducibile*.
Definiamo un po' di termini: l'**ordine** di un campo finito è il numero dei suoi elementi, la **caratteristica** di un campo finito è il numero di volte che dobbiamo sommare un elemento a *0* prima di ottenere di nuovo *0*, in un campo di *ordine* p^k la *caratteristica* è p. Se la somma non raggiunge mai lo *0*, cosa che accade nei campi infiniti, allora la *caratteristica* è *0*.
Chiusura Algebrica: un campo F si dice **algebricamente chiuso** se ogni polinomio univariato (di ordine maggiore di *0*, cioè non costanti) ha uno zero in F.

6.2 Curva Elittica

Consideriamo **EC** già poste nella forma normalizzata di ***Weierstrass***, ovvero diremo che una curva elittica **E** su un campo F è una curva cubica su F definita dall'equazione:

$$y^2 = x^3 + a \cdot x + b, \qquad a, b \in F$$

Se $(\overline{x}, \overline{y})$ soddisfano l'equazione allora anche $(\overline{x}, -\overline{y})$ soddisfa l'equazione $\Rightarrow$ **simmetrico** rispetto all'**asse x**.

$$P = (\overline{x}, \overline{y}) \quad \Rightarrow \quad -P = (\overline{x}, -\overline{y})$$

In crittografia sono di interesse le **curve lisce**, anche dette **smooth**, avvero curve in cui le derivate parziali non si annullano simultaneamente $\Rightarrow$ in ogni punto esiste una e una sola tangente:

$$\begin{aligned} \frac{d}{dy}E(x,y) &= \frac{d}{dy}(y^2 - x^3 - ax - b) = 2y \\ &\Rightarrow \text{ che si annulla solo per } y = 0 \\ \frac{d}{dx}E(x,y) &= \frac{d}{dx}y^2 - x^3 - ax - b = 3x^2 + a \end{aligned}$$

Avremo che i punti che rendono la **curva non smooth** soddisfano **simultaneamente**:

$$\begin{cases} x^3 + ax + b = 0 \\ 3x^2 + a = 0 \end{cases}$$

Se poniamo $x = 0$ si vede che la curva **non è liscia** se e solo se $a = b = 0$, se $a = 0, b \neq 0$ le due equazioni non si possono annullare contemporaneamente e quindi la curva sarà **smooth**. Avremo che, ovviamente, nel caso in cui $y^2 = x^3$ la curva non sarà liscia. In conclusione le curve non lisce vanno cercate ponendo $a < 0 \wedge x \neq 0$.

$$\begin{cases} x^3 + ax + b = 0 \\ 3x^2 + a = 0 \end{cases} \Rightarrow \begin{cases} 3 \cdot (x^3 + ax + b) = 3 \cdot 0 \\ x \cdot (3x^2 + a) = x \cdot 0 \end{cases} \Rightarrow \begin{cases} 3x^3 + 3ax + 3b = 0 \ (-) \\ 3x^3 + ax = 0 \end{cases} \Rightarrow$$

$$\begin{cases} 2ax + 3b = 0 \\ 3x^2 + a = 0 \end{cases} \Rightarrow \begin{cases} x = -\frac{3b}{2a} \\ \frac{27b^2 + 4a^3}{4a^2} = 0 \end{cases}$$

Avremo quindi ottenuto che la curva **non è liscia** se e solo se la quantità $\Delta = 27b^2 + 4a^3$ è nulla, questa quantità viene anche definita **discriminante della curva**, e nel caso avremo il punto di **singolarità** in $P(-\frac{3b}{2a}, 0)$. Normalmente una **EC** nella forma di *Weierstrass* è definita dai parametri (a, b) e quindi si indica con $E_{(a,b)}$.

6.2.1 Curve Elittiche Smooth

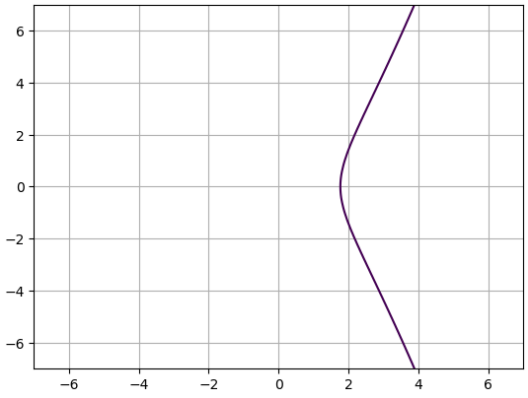


Figura 6.1: $E_{(-2,-2)}$

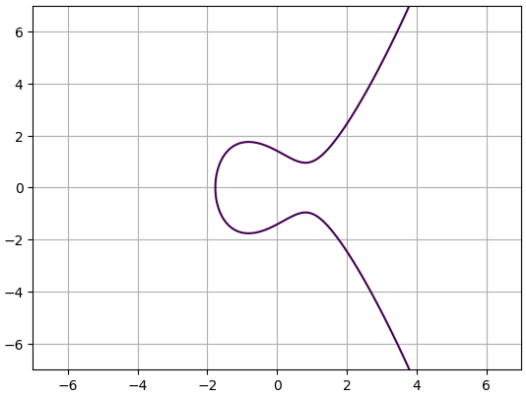


Figura 6.2: $E_{(-2,2)}$

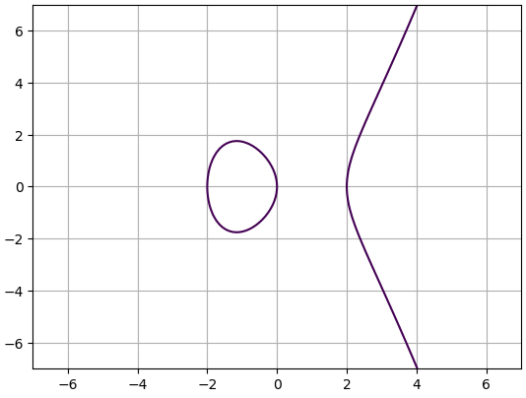


Figura 6.3: $E_{(-4,0)}$

6.2.2 Curve Elittiche non Smooth

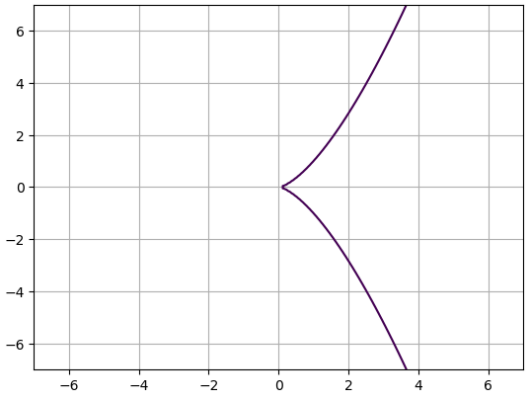


Figura 6.4: $E_{(0,0)}$

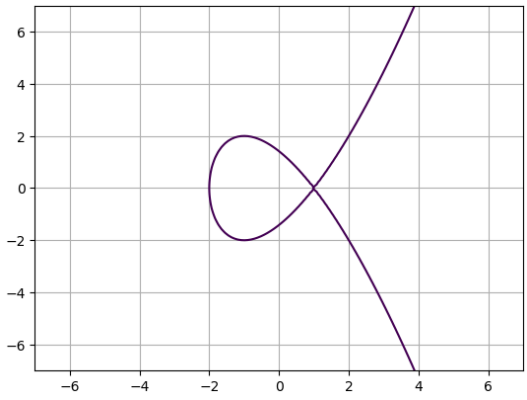


Figura 6.5: $E_{(-3,2)}$

6.3 Operazioni Geometriche su una Curva

Intersezioni con una retta: definiamo R una retta di equazione $y = mx + q$ e una curva E di equazione $y^2 = x^3 + ax + b$, avremo quindi le coordinate x di intersezione tra R e E ($E \cup R$) definite implicitamente dagli zeri di $E(x, y) = y^2 - x^3 - ax - b$.

$$\begin{aligned} (mx+q)^2 &= x^3 + ax + b \\ mx^2 + q^2 + 2mxq &= x^3 + ax + b \\ x^3 + ax + b - mx^2 - q^2 - 2mxq &= 0 \\ E(x,y) &= x^3 - mx^2 - (a - 2mq) \cdot x - q^2 \end{aligned}$$

Si tratta di un polinomio **cubico** e sappiamo che, nel campo reale, esso deve avere **uno** o **tre zeri reali** (se ne ha uno solo, gli altri due zeri sono **complessi coniugati**).
Andando a studiare i casi particolari avremo:

$$E(x,y) = \begin{cases} x^3 + ax + b - q^2 & m = 0 \\ y^2 - c^3 - ac - b & R : x = c \end{cases}$$

Su un **campo finito** $\boldsymbol{F}$ non è assolutamente detto che un'equazione $P(x) = 0$, dove P è un polinomio a coefficienti in F di **grado dispari**, abbia almeno una soluzione in F. È importante notare che nel caso in cui $F = \mathbb{Z}_p$ ha due zeri in F, allora anche il

terzo zero è in F, tenendo presenti le molteplicità degli zero. Nel caso in cui, invece il polinomio a grado 2, ovvero è una **retta verticale** che incrocia la curva in **due punti** sarà necessario considerare un ulteriore punto di intersezione, che apparterrà ad un'estensione di F (reale o finito).

$m = 0 \Rightarrow$ rette orizzontali

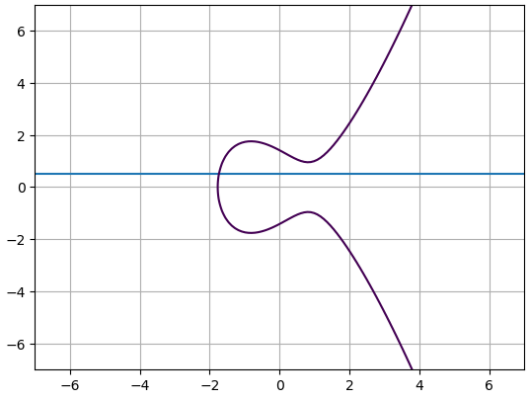


Figura 6.6: $E_{(-2,2)} \cup q = 0.5$

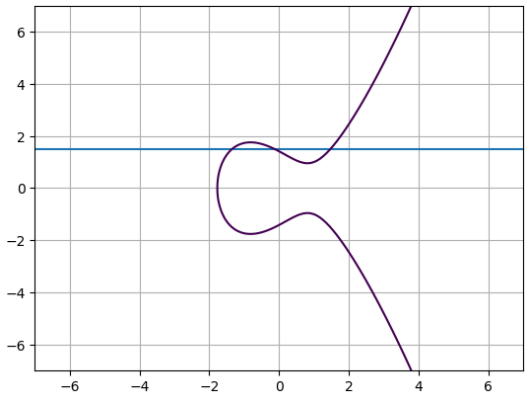


Figura 6.7: $E_{(-2,2)} \cup q = 1.5$

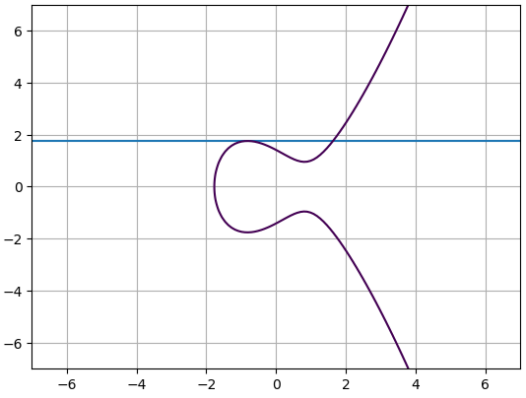


Figura 6.8: $E_{(-2,2)} \cup q = 1.7427$

- Nella *Figura 6.6* abbiamo che la retta interseca una sola volta la curva.
- Nella *Figura 6.7* abbiamo che la retta interseca la curva in tre punti, tutti con molteplicità *1*.
- Nella *Figura 6.8* abbiamo che la retta interseca 2 volte la curva, il primo punto altro non è che il punto di intersezione tra la curva e la tangente della curva in quel punto, che quindi avrà molteplicità *2* e l'altro invece appartiene alla curva e quindi ha molteplicità *1*

$m = \infty \Rightarrow$ rette verticali

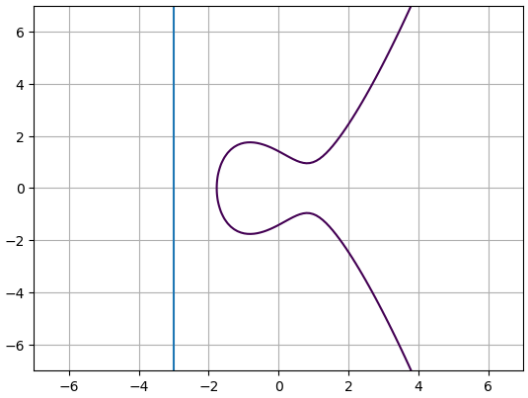


Figura 6.9: $E_{(-2,2)} \cup m = \infty, q = -3$

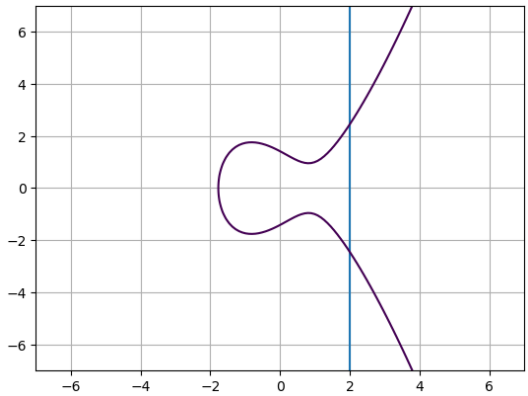


Figura 6.10: $E_{(-2,2)} \cup m = \infty, q = 2$

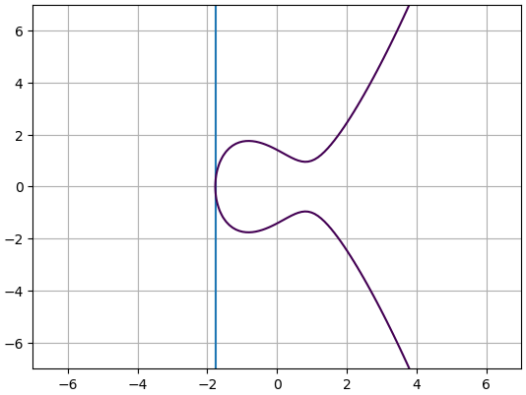


Figura 6.11: $E_{(-2,2)} \cup m = \infty, q = -1.76$

- Nella *Figura 6.9* abbiamo che la retta non interseca la curva.
- Nella *Figura 6.10* abbiamo che la retta interseca la curva in due punti, tutti con molteplicità *1*.
- Nella *Figura 6.11* abbiamo che la retta interseca la curva come tangente di essa, quindi il punto avrà molteplicità *2*.

$m > 0$, *finito* $\Rightarrow$ rette oblique

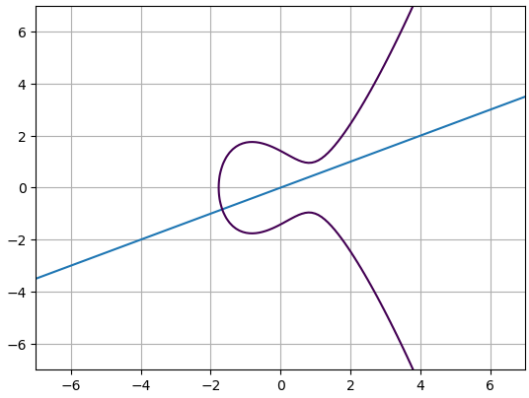


Figura 6.12: $E_{(-2,2)} \cup m = 4$

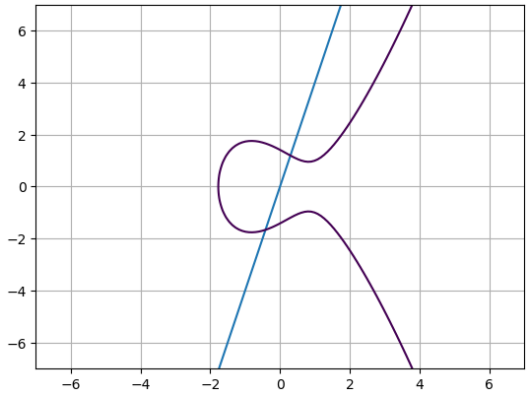


Figura 6.13: $E_{(-2,2)} \cup m = 0.5$

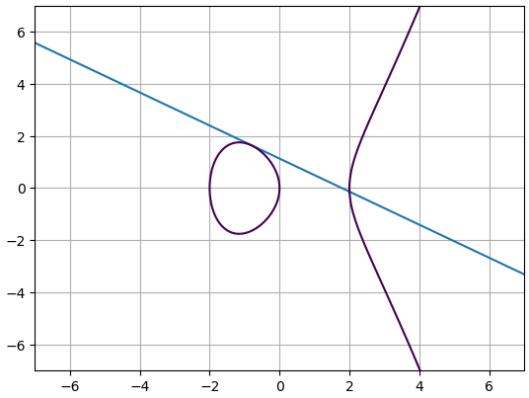


Figura 6.14: $E_{(-2,2)} \cup m = -0.741, q = 1.416$

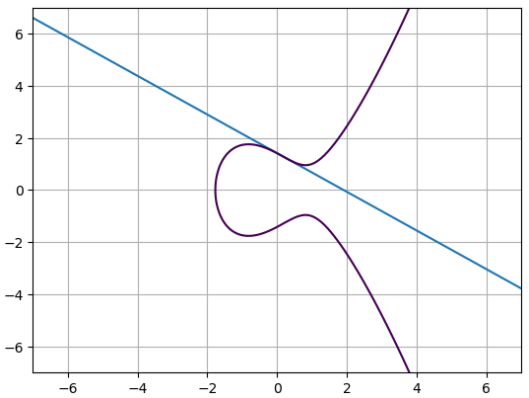


Figura 6.15: $E_{(-2,2)} \cup m = -0.634, q = 1.132$

- Nella *Figura 6.12* avremo che la retta interseca la curva in un unico punto.
- Nella *Figura 6.13* in questo caso avremo che la retta interseca, visivamente, la curva in soli due punti, entrmabi con molteplicità *1*, ma sappiamo che siccome la curva E, al crescere di x, si comporta come $x^{\frac{3}{2}}$, quindi con velocità *superlineare*, sappiamo che sarà presente un altro punto di intersezione con molteplicità *1*.
- Nella *Figura 6.14*: abbiamo che la retta interseca 2 volte la curva, il primo punto altro non è che il punto di intersezione tra la curva e la tangente della curva in quel punto, che quindi avrà molteplicità *2* e l'altro invece appartiene alla curva e quindi ha molteplicità *1*

- Nella *Figura 6.15*: in questo caso la retta interseca la curva in un unico punto, che però essendo un punto di **flesso** della curva ha molteplicità *3*.

Abbiamo definito che ai fini **crittografici** i casi di interesse sono quelli in cui sono presenti **3 punti di intersezione**. È infatti possibile definire un **gruppo** costituito proprio dai punti sulla curva. Nel caso di rette **verticali** ovvero con $m = \infty$ sono presente unicamente due punti di intersezione sulla curva. Detta in maniera discorsiva, vorremmo che quando sono presenti due intersezioni ce ne fosse **sempre** una terza, mentre invece se non ci sono intersezioni o ce n'è una sola **non sono rilevanti**.
La soluzione più semplice è quella di considerare un punto "extra", che indicheremo con $\mathcal{O}$, detto anche **Punto all'Infinito**. Il punto viene aggiunto ad F ed è inteso far parte di qualsiasi curva E su F. Come vedremo $\mathcal{O}$ fungerà da **elemento neutro** dell'operazione di gruppo.
Proprietà di $\mathcal{O}$:

- qualsiasi **retta verticale** interseca E in $\mathcal{O}$ con molteplicità *1*, ne consegue che se R ha già due intersezioni con E il **punto all'infinito** costituirà il terzo punto di intersezione.

- nessuna retta con di equazione $y = mx + q$ con $m > 0$ finito interseca il punto $\mathcal{O}$.

- nel punto $\mathcal{O}$ la curva ha tangente *t*. Si suppone che che t abbia come unica intersezione con la curva proprio il punto $\mathcal{O}$ con molteplicità *3*.

- $\mathcal{O}$ coincide con il suo opposto $-\mathcal{O}$.

- con l'introduzione del punto all'infinito, E gode della seguente proprietà: **Sia R una retta definita in F, se R ha due punti di intersezione in F con la curva E, allora ha anche un terzo punto di intersezione in F**.

Addizione su Curva Elittica

La somma di due punti $A, B \in E$ è definita come $A + B = -C$ ovvero la somma di A e B è il punto simmetrico al terzo punto di intersezione che risolve il sistema:

$$\begin{cases} y^2 = x^3 + ax + b \\ y = mx + q \end{cases}$$

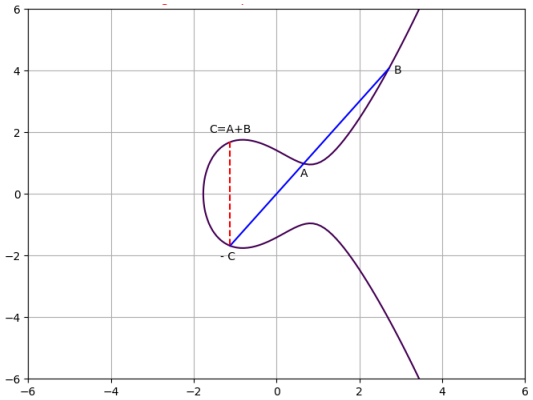


Figura 6.16: Caso generale

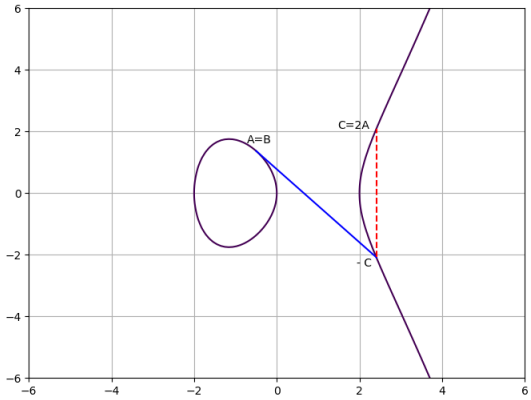


Figura 6.17: Caso di punti coincidenti

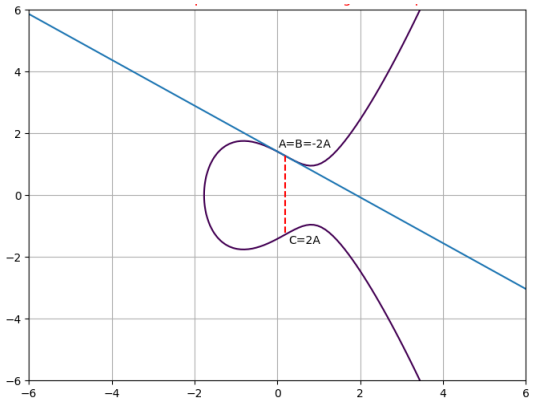


Figura 6.18: Caso di punto con flesso

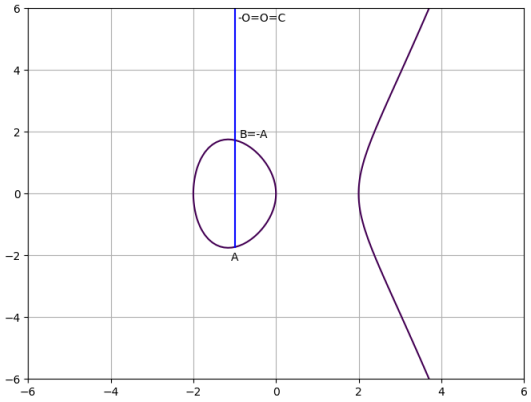


Figura 6.19: Caso retta verticale

I punti sulla curva elittica con operazione di addizione appena definita formano un gruppo **abelliano**, ovvero un gruppo dove è definita la proprietà **commutativa**. Il gruppo generato mantiene anche la proprietà di **associazione** rispetto alla somma di punti.

1. **Chiusura**, in forza anche dell'introduzione del punto $\mathcal{O}$.

   - i casi in cui i due punti sono diversi da $\mathcal{O}$ sono definiti geometricamente nelle *Figure 6.16, 6.17, 6.18* in tutti i casi $A + B$ è ancora un punto della curva (eventualmente $\mathcal{O}$).

- $\mathcal{O} + \mathcal{O} = \mathcal{O}$, infatti per la proprietà del **punto all'infinito** sappiamo che $\mathcal{O}$ ha molteplicità 3 e che quindi la terza intersezione è $\mathcal{O}$, ma anche il suo **simmetrico**.
- $A + \mathcal{O} = A$.

2. $\mathcal{O}$ è l'elemento **neutro**: $A + \mathcal{O} = \mathcal{O} + A = A$
3. ogni elemento A ha un opposto, precisamente il punto di simmetria sulla curva -A, infatti $A + (-A) = \mathcal{O}$.
4. la proprietà **associativa** è verificata.
5. la proprietà **commutativa** è verificabile geometricamente in quanto la retta che passa per due punti è individuabile indipendentemente dall'ordine dei punti.

Caso A $\neq$ B

Noti i punti $A(x_a, y_a)$ e $B(x_b, y_b)$ che sono soluzioni per il sistema:

$$\begin{cases} y^2 = x^3 + ax + b \\ y = mx + q \end{cases} \Rightarrow R = \begin{cases} y = y_a = y_b \qquad \text{se } m = 0 \\ y = (\frac{y_b - y_a}{x_b - x_a})x + (y_a - (\frac{y_b - y_a}{x_b - x_a}) \cdot x_a) \end{cases}$$

$$\Rightarrow (mx + q)^2 = x^3 + ax + b \Rightarrow x^3 - mx^2 + (a - 2mq)x + b - q^2 = 0$$

Siccome due soluzione le conosciamo già e sappiamo che sono $A(x_a, y_a)$ e $B(x_b, y_b)$, possiamo visualizzare le x del polinomio di terzo grado come: $(x - x_a)(x - x_b)(x - x_{\overline{c}}) = 0$ a noi interessa trovare il punto $-C(x_{\overline{c}}, y_{\overline{c}})$ che intersechi la curva E appartente alla retta descritta da A e B.

$$(x - x_a)(x - x_b)(x - x_c) = 0$$

$$x^3 - mx^2 + (a - 2mq)x + b - q^2 = 0$$

$$(x - x_a)(x - x_b)(x - x_c) = x^3 - mx^2 + (a - 2mq)x + b - q^2$$

$$x^3 + x^2 \cdot (-x_a - x_b - x_c) + x \cdot (x_a x_b + x_a x_c + x_b x_c) + x_a x_b x_c = x^3 - mx^2 + (a - 2mq)x + b - q^2$$

Ugualiando i coefficienti di grado *2* avremo: $-m^2 = -x_a - x_b - x_c$ e quindi potremo trovare la coordianta del punto $-C(x_{\bar{c}}, y_{\bar{c}}) = (m^2 - x_a - x_b, m \cdot x_{\bar{c}} + q)$. Il punto somma $C = A + B = -(-C)$ e siccome abbiamo detto che la **EC** è simmetrica rispetto all'asse delle x avremo che $x_{\bar{c}} = x_c$ e $y_{\bar{c}} = -y_c$ andando così a descrivere il punto $C(x_c, y_c)$.

Caso A = B

Quando i due punti coincidono è necessario calcolare il **coefficente angolare *m*** della retta R_A tangente alla curva per il punto A. Anche se la curva non è definita in maniera esplicita è possibile rappresentarla come l'unione dei graifici di due funzioni, più precisamente:

$$y_1(x) = \sqrt{x^3 + ax + b} \quad \cup y_2(x) = -\sqrt{x^3 + ax + b}$$

Dobbiamo quindi calcolare la derivata $\frac{d}{dx} y_1(x')$ e $\frac{d}{dx} y_2(x')$ dove (x', y') è un punto definito sulla curva y_1 se $y' \geq 0$, quindi vale $y_1(x') = y'$ mentre invece è definito sulla curva y_2 se $y' \leq 0$, quindi vale $y_2(x') = y'$ possiamo calcolare i coefficienti delle rette tangenti:

$$\frac{d}{dx} y_1(x') = \frac{3x'^2 + a}{2\sqrt{x'^3 + ax' + b}} = \frac{3x'^2 + a}{2y_1(x')} = \frac{3x'^2 + a}{2y'}$$
$$\frac{d}{dx} y_2(x') = -\frac{3x'^2 + a}{2\sqrt{x'^3 + ax' + b}} = -\frac{3x'^2 + a}{2(-y_2(x'))} = -\frac{3x'^2 + a}{-2y'} = \frac{3x'^2 + a}{2y'}$$

Avremo quindi che il calcolo del **coefficente angolare** è $m = \frac{3x'^2 + a}{2y'}$.

Pseudo-codice per il calcolo di **C = A + B**:

```
def sumPointOnEC(A: point, B: point) -> point:
    if A == O: return B
    if B == O: return A
    if B == -A: return O

    if A ≠ B: m = (y_b − y_a)/(x_b − x_a)
    if A == B: m = (3x_a^2 + a)/(2y_a)

    # C(x, y)
    return point(m^2 − x_a − x_b, −(mx_c + q))
```

6.4 Curve in $\mathbb{Z}_p$

Indicheremo una curva elittica a coefficienti a e b, definita sul campo $\mathbb{Z}_p$, usando la notazione $E_{a,b}(\mathbb{Z}_p)$.

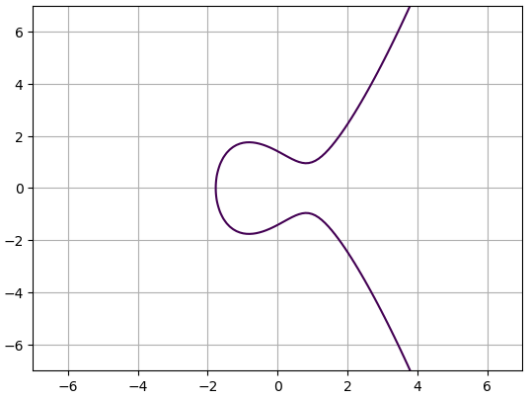


Figura 6.20: $E_{(-2,-2)}$

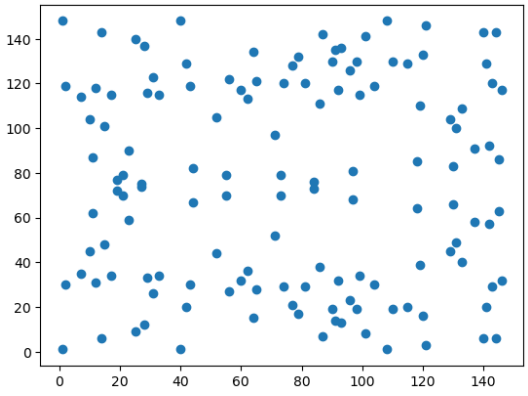


Figura 6.21: $E_{-2,2}(\mathbb{Z}_{149})$

Anche se, come possiamo notare, la geometria qui non ci può aiutare, i risultati algebrici rimangono ancora validi. Se un'equazione cubica (a coefficienti in $\mathbb{Z}_p$) ha due radici in $\mathbb{Z}_p$, allora ha anche la terza radice in $\mathbb{Z}_p$. Questo consenti di definire l'operazione di addizione come nel caso reale. Analogamente a quanto fatto per il gruppo $\mathbb{Z}_p^*$, possiamo definire dei **sottogruppi ciclici** dei punti $S_{E_{a,b}}(\mathbb{Z}_p)$. Il sottogruppo generato da un punto $g(x_g, y_g)$ è definito nel seguente modo:

$$S_{E_{a,b}(\mathbb{Z}_p)}(g) = \{a \in E_{a,b}(\mathbb{Z}_p) | \exists k \geq 0, a = k \cdot g = \underbrace{g + g + \ldots + g}_{k-1 \text{ somme}}\}$$

L'operazione $k \cdot g$, dove k è un intero non negativo e g è un punto sulla curva, è detta ***moltiplicazione scalare***. Normalmente il gruppo dei punti sulla curva **non è ciclico**, per esso vale il teorema di ***lagrange***, ma non quello fondamentale dei gruppi ciclici.

Logaritmo Discreto su Curva Elittica

Se g è un punto sulla curva $S_{E_{a,b}(\mathbb{Z}_p)}$ che genera l'intero gruppo (quindi g è una **radice primitiva** di $S_{E_{a,b}(\mathbb{Z}_p)}$), allora per ogni altro punto $z \in S_{E_{a,b}(\mathbb{Z}_p)} \rightarrow \exists k \geq 0 \mid z = k \cdot g$. Il minimo intero k che soddisfa questa ugualianza è il **logaritmo a base *g* di *z*** rappresentabile nella forma $k = \log_g z$. Da un punto di vista computazionale, dati

k e g è facile trovare il valore di z, infatti l'algoritmo segue lo stesso schema dell'esponenziale modulare, ovvero il ***Recursive Doubling***. Il calcolo di z parte dalla rappresentazione binaria di k accumulando le quantità solo in corrispondenza degli *1* della rappresentazione binaria di k, il calcolo inverso, ovvero dati g e z trovare k è invece **computazionalemente difficile** e non sono noti algoritmi con costo polinomiale.
Date le proprietà presenti del gruppo $E_{a,b}(\mathbb{Z}_p)$ è quindi facile immaginare del perché le **EC** abbiamo suscitato un tale interesse nell'ambito della crittografia. È però interessante osservare la questione dei **numeri di punti sulla curva**, infatti questo è l'**ordine** del nostro gruppo $E_{a,b}(\mathbb{Z}_p)$, quando si lavorava su gruppo $\mathbb{Z}_p^*$ l'ordine veniva facilmente calcolato dalla quantità $p-1$ e quindi si potevano prendere delle precauzioni per evitare che i sottogruppi di $\mathbb{Z}_p^*$ avessero un ordine molto piccolo, grazie al **teorema di Lagrange** e la diretta costruzione del primo del tipo $p = 2q + 1$ dove anche q fosse un numero primo. In questo modo si poteva controllare l'ordine dei sottogruppi di $\mathbb{Z}_p^*$ e quindi tenerli controllati in modo da evitare che fossero fattori semplici di p (concetto dei **safe prime**). Nel caso delle **EC** non abbiamo un "teorema" che ci possa guidare alla scelta dei parametri *a, b, p* in modo da controllare le dimensioni del gruppo dei punti sulla curva.

Numeri di Punti sulla Curva: se p è il modulo, il numero di punti su una curva può superare il valore di $2p+1$ questo perché se $x^3 + ax + b \bmod p \neq 0$ è un residuo quadratico, allora al valore x corrispondono due radici in p e quindi **due punti sulla curva**. Tuttavia per raggiungere il limite $2p + 1$ è necessario che $x^3 + ax + b$ sia un residuo quadratico modulo p $\forall x \in \{0, 1, ..., p - 1\}$ è però ragionevole attendersi che circa metà dei valori siano residui quadrati e metà residui non-quadratici in questo modo il numero di punti torna ad essere circa p. Il numero di punti su una curva viene rappresentato da: $\#E(a, b, p)$.
Il ***Teorema di Hasse*** fornisce un *bound* più preciso:

$$|\#E(a, b, p) - (p + 1)| \leq 2\sqrt{p}$$

Dal punto di vista algoritmico, il miglior risultato noto per determinare il numero "esatto" di punti è dovuto a ***René Schoof*** la cui versione non ottimizzata dell'algoritmo ha complessità $\mathcal{O}((\log p)^8)$ va però tenuto presente che per ogni curva l'algorit-

mo va eseguito una sola volta. In generale è sconsigliato cercare delle **safe curves** in autonomia, esistono già dei database in cui vengono salvate curve ritenute sicure (https://safecurves.cr.yp.to/).

Capitolo 7

Algoritmi di Crittografia su elittica

I parametri pubblici in un protocollo basato su una curva elittica includono sempre i seguenti dati:

1. la **curva**: e quindi tutti i paramentri che la caratteristicano, ovvero a, b (i parametri dell'equazione) e p (il modulo) e il numero di **punti** sulla curva.
2. un punto P che è generatore di un **sottogruppo ciclico** del gruppo formato da tutti i punti della curva (possibilmente di ordine elevato).
3. un secondo punto Q formato da $Q = k \cdot P$, per un qualche intero k che è il segreto della comunicazione.

È chiaro che il poter risalire a k da parte di *Eve* permette di violare il codice, tuttavia il risalire a k dai parametri pubblici dell'algoritmo è definito ***Elliptic Curve Discrete Logarithm Problem*** anche noto come ***ECDLP*** ed è computazionalmente intrattabile. Tuttavia è possibile risalire ad una **collisione** per poter risalire a k dati i valori pubblici dell'algoritmo, attraverso la combinazione lineare a coefficienti interi dei due punti P e Q.
In altri termini dobbiamo trovare due coppie di numeri α_p, α_q e β_p, β_q tali che valga l'equazione:

$$\begin{aligned} \alpha_p \cdot P + \alpha_q \cdot Q &= \beta_p \cdot P + \beta_q \cdot Q \\ \alpha_p \cdot P + k \cdot \alpha_q \cdot P &= \beta_p \cdot P + k \cdot \beta_q \cdot P \;||\; \Leftarrow Q = k \cdot P \\ (\alpha_p + k \cdot \alpha_q)P &= (\beta_p + k \cdot \beta_q)Q \end{aligned}$$

Analizzando i passaggi: i valori che cerca *Eve* devono essere tali che $\alpha_p + k \cdot \alpha_q \equiv 0 \bmod \#E$ e $\beta_p + k \cdot \beta_q \equiv 0 \bmod \#E$ la ragione va ricercata utilizzando il **teorema di Lagrange**, infatti se i coefficenti sono congrui modulo il numero di punti, saranno congrui anche all'ordine del sottogruppo generato da P in quanto, un gruppo che ha ordine k avrà come ordine dei sottogruppi dei **divisori** di k, se dunque quale la congruenza:

$$\begin{aligned}
\alpha_p + k \cdot \alpha_q &\equiv 0 \bmod \#E \\
\beta_p + k \cdot \beta_q &\equiv 0 \bmod \#E \\
\alpha_p + k \cdot \alpha_q &\equiv \beta_p + k \cdot \beta_q \bmod \#E \\
\alpha_p + k \cdot \alpha_q - \beta_p - k \cdot \beta_q &\equiv 0 \bmod \#E \\
\alpha_p - \beta_p - k \cdot (\beta_q - \alpha_q) &\equiv 0 \bmod \#E \\
\alpha_p - \beta_p &\equiv k \cdot (\beta_q - \alpha_q) \bmod \#E
\end{aligned}$$

Ottenendo quindi:

$$k = (\alpha_p - \beta_p) \cdot (\beta_q - \alpha_q)^{-1} \bmod \#E \iff gcd(\beta_q - \alpha_q, \#E) = 1$$

Con un argomento riducibile all "dimostrazione" del **paradosso del compleanno** possiamo dire che il tempo atteso nel caso di numeri di n bit è $\mathcal{O}(2^{\frac{n}{2}})$.

I **protocolli crittografici** più comunemente utilizzati, fra quelli che impiegano curve elittiche, riguardano lo **scambio di chiavi** e la **firma digitale**. È infatti possibile avere con le **EC** un grado di sicurezza (espresso in bit) di $\frac{n}{2}$, con chiavi di n bit e dunque $n = 256$ è tipicamente una lunghezza sufficente, quindi molto "performante", ma insufficiente per la **cifratura di messaggi**.

7.1 Elliptic Curve Diffie-Hellman

Descrizione del Protocollo:

1. *Alice* e *Bob* si accordano su una **curva elittica** $\boldsymbol{E}$ $(E_{a,b}(\mathbb{Z}_p), \#E)$ e su un punto $P \in E_{a,b}(\mathbb{Z}_p)$.
2. *Alice* sceglie a caso un numero k_a e determina il punto $A = k_a \cdot P$ che invia a *Bob*.
3. *Bob* sceglie a caso un numero k_b e determina il punto $B = k_b \cdot P$ che invia a *Alice*.
4. *Alice* calcola $Z_a = k_a \cdot B$ con il punto B ricevuto da *Bob*
5. *Bob* calcola $Z_b = k_b \cdot A$ con il punto A ricevuto da *Alice*
6. il punto $Z = Z_a = Z_b$ è il **segreto condiviso**, normalmente siccome Z è nella forma $Z(x_z, y_z)$ viene scelta una delle due coordinate come segreto.

La **correttezza del protocollo** è una l'applicazione della proprietà **associativa** all'interno di un gruppo $E_{a,b}(\mathbb{Z}_p)$:

$$Z_a = k_a \cdot B = k_a \cdot (k_b \cdot P) = k_b \cdot (k_a \cdot P) = k_b \cdot A = Z_b$$

Per quanto riguarda invece l'**efficienza** le operazioni più costose sono chiaramente i "prodotti scalari" per il calcolo della "chiave pubblica" e del segreto condiviso, dove però viene utilizzato l'algoritmo di **recursive doubling** e quindi possono essere considerati come addizioni di punti che è lineare alla lunghezza delle costanti segrete in gioco. Ogni addizione richiede, perciò, *4* o *5* moltiplicazioni nel campo $\mathbb{Z}_p^+$ oltre alle riduzioni in modulo. Il costo computazionale finale è perciò $\mathcal{O}(n^3)$ bit-operation.

Attacco basato sull'uso di una differente curva "debole"

La **condizione** per l'attacco è che *Bob* non esegua un controllo sul punto che *Alice* (*Eve*) invia come **chiave pubblica**.

Definiamo $E_{a,b}(\mathbb{Z}_p)$ come curva scelta per la comunicazone, $\#E$ il numero di punti presenti sulla curva e un punto $P \in E_{a,b}(\mathbb{Z}_p)$. *Bob* sceglie la propria **chiave privata** k_b e calcola il punto $B = k_b \cdot P$ e lo invia ad *Eve*.

Eve sceglie un punto $A \in \overline{E}_{a,b}(\mathbb{Z}_p) \mid \mathcal{O} = p \cdot A$ dove avremo che $\#\overline{E} = p \cdot q$. Definiamo $\overline{E}_{a,b}(\mathbb{Z}_p)$ **"curva debole"** e $\#\overline{E}$ il numero di punti presenti sulla curva. A quel punto invia a *Bob* la propria chiave pubblica: A. *Bob* calcolerà il **segreto condiviso** come $Z = k_b \cdot A$ e potrà iniziare a inviare messaggi cifrati ad *Eve* come $C = Enc(M, hash(Z))$.

Una volta ottenuto un messaggio cifrato *Eve* non riuscirà a decifrarlo nella maniera *standard*, però per costruzione potrà ricercare un punto $I \in S_{\overline{E}_{a,b}(\mathbb{Z}_p)}(A) \mid I = (A \cdot \mathcal{O}) \cdot m,\ m \in \{0, 1, ..., p-1\}$ che permetterà di decifrare il messaggio $M = Dec(C, hash(I))$.

A questo punto I è quindi associato un valore m che altro non è che il resto della divisione di k_b ovvero il segreto di *Bob* per uno dei due fattori di $\#\overline{E} \to p$.

Una volta ottenuto il primo dei due parametri per riuscire ad ottenere in maniera illegittima la **chiave segreta** di *Bob*, *Eve* dovrà "fingere" di aver sbagliato ad inviare la propria **chiave pubblica** attraverso tecniche di ***Social Engineering*** e reinviare a *Bob* un punto $A' \in \overline{E}_{a,b}(\mathbb{Z}_p) \mid \mathcal{O} = q \cdot A'$ dove q è l'altro fattore di $\#\overline{E}$, *Bob* ripeterà i passaggi fino a che *Alice* non dovrà decifrare il nuovo messaggio inviato da *Bob* utilizzando la formula: $I \in S_{\overline{E}_{a,b}(\mathbb{Z}_p)}(A) \mid I = (A \cdot \mathcal{O}) \cdot r,\ r \in \{0, 1, ..., q-1\}$ una volta che avra ottenuto il punto I che gli permetterà di decifrare il nuovo messaggio $M = Dec(C, hash(I))$ e quindi il corrispondente r che altro non è che il resto della divisione intera di k_b ovvero il segreto di *Bob* per uno dei due fattori di $\#\overline{E} \to q$.

A questo punto *Eve* può usare il ***Chinese Remainder Theorem*** per costruire un $C \equiv k_b \bmod \#\overline{E}$

$$C = q \cdot (q^{-1} \bmod p) \cdot m + p \cdot (p^{-1} \bmod q) \cdot r$$
$$k_b = C \bmod \#\overline{E}$$

Vista la parte di ***Social Engineering*** si cerca di contenere il numero di "errori di invio della chiave" il più basso possibile $\#\overline{E} = p \cdot q$, ma generalizzando, potrebbe essere $\#\overline{E} = p_1 \cdot p_2 \cdot p_h$ e bisognerebbe ingannare *Bob* $h - 1$ volte.

7.2 Elliptic Curve - Digital Signature Algorithm

Descrizione del Protocollo:

- **Generazione delle Chiavi: *Alice***
  1. fissa la curva $E_{a,b}(\mathbb{Z}_p)$ e un punto base $P \in E_{a,b}(\mathbb{Z}_p)$.
  2. determinato un numero $N = \#E_{a,b}(\mathbb{Z}_p)$ di punti sulla curva genera a caso un elemento $a \in \mathbb{Z}_p^+$ che conserva come **chiave privata**.
  3. calcola $A = a \cdot P$ e rende noto $(E_{a,b}(\mathbb{Z}_p),\ A)$ come chiave **chiave pubblica**.
- **Firma del Messaggio: *Alice***
  1. applica una funzione **hash** (pubblicamente nota) al messaggio M da firmare, generando $h = hash(M) \bmod N$.
  2. sceglie a caso un numero $k \mid 0 < k < N \wedge gcd(k, N) = 1$ e calcola il punto $Q = k \cdot P$ sulla curva, $Q(x_q, y_q)$.
  3. Pone $r = Q_x \bmod N$ e $s = k^{-1} \cdot (h + r \cdot a) \bmod N$
  4. invia la $(M, (r, s))$.
- **Verifica della Firma: *Bob***
  1. calcola l'**inverso moltiplicativo** di s modulo N cioè $w = s^{-1} \bmod N = k \cdot (h + r \cdot a)^{-1} \bmod N$.
  2. calcola i due prodotti
  $$u = hash(M) \cdot w$$
  $$v = r \cdot w$$
  3. determina il punto $R = (uP + vA) \bmod N$.
  4. accetta solo se $x_r = r$

La **correttezza** dell'algoritmo è verificata in quanto (i calcoli sono modulo N):

$$\begin{aligned} R &= u \cdot P + v \cdot A \\ &= (h \cdot w) \cdot P + (r \cdot w) \cdot A \\ &= h \cdot w \cdot P + r \cdot w(a \cdot P) \\ &= w \cdot (h + r \cdot a) \cdot P \\ &= (k \cdot (h + r \cdot a)^{-1}) \cdot (h + r \cdot a) \cdot P \\ &= k \cdot P \\ &= Q \end{aligned}$$

Il valore casuale k deve essere usato una sola volta, altrimenti è vulnerabile. Supponiamo che due messaggi M_1 e M_2 con corrispondenti valori di hash h_1 e h_2 vengano firmati con lo stesso valore k. Questo porta ad avere anche lo stesso valore di r, che è la coordinata x (modulo N) del punto $Q = k \cdot P$.
Indichiamo con (r, s_1) e (r, s_2) le due firme, l'attaccante può calcolare la quantità:

$$\begin{aligned} s_1 - s_2 &= k^{-1} \cdot (h_1 + r \cdot a) - k^{-1} \cdot (h_2 + r \cdot a) \\ &= (h_1 + r \cdot a - h_2 - r \cdot a) \cdot k^{-1} \\ &= (h_1 - h_2) \cdot k^{-1} \end{aligned}$$

Riuscendo ad ottenere:

$$k = (s_1 - s_2)^{-1} \cdot (h_1 - h_2) \bmod N$$

Una volta ottenuto il k per risalire alla chiave privata bisogna utilizzare una delle due firme:

$$\begin{aligned} s_1 &= k^{-1} \cdot (h_1 + r \cdot a) \\ k \cdot s_1 &= (h_1 + r \cdot a) \\ k \cdot s_1 - h_1 &= r \cdot a \\ (k \cdot s_1 - h_1) \cdot r^{-1} &= a \end{aligned}$$