

Appunti di Crittografia A.A.2020-2021

(basato sulla dispensa di Aleandro Prudenzano)

Gabriele Frassi (**g.frassi2@studenti.unipi.it**)

11 febbraio 2022

FORSE LO AVETE FATTO A MATEMATICA DISCRETA (cit. Bernasconi)

Indice

1 Alcuni dettagli sul corso	9
I Introduzione	10
2 Introduzione alla crittologia	11
2.1 Crittografia e crittoanalisi	11
2.2 Scenario tipico della crittografia	11
2.2.1 Funzione di cifratura (criptaggio)	11
2.2.2 Funzione di decifratura (decifrazione)	12
2.2.3 Schema di comunicazione	12
2.3 Esempi antichi	12
2.4 Livello di segretezza	13
2.4.1 Ridefinizione delle funzioni di criptaggio e decifrazione	13
2.5 Crittoanalisi	14
2.5.1 Tipologie di attacchi	14
2.5.2 Attacchi man-in-the-middle	14
3 Rimandi alla teoria dell'informazione	15
3.1 Rappresentazione matematica di oggetti	15
3.1.1 Rappresentazione degli oggetti con l'alfabeto	15
3.1.1.1 Alfabeto unario	15
3.1.1.2 Alfabeto binario	16
3.1.1.3 Alfabeto s -ario	16
3.1.2 Rappresentazione di interi	17
3.2 Differenza tra calcolabilità e complessità	17
3.3 Teoria della calcolabilità	17
3.3.1 Problemi computazionali	17
3.3.2 Esistenza di problemi non decidibili	18
3.3.2.1 Definizione: insiemi con stessa cardinalità	18
3.3.2.2 Definizione: insieme numerabile	18
3.3.2.3 Dimostrazione	19
3.3.2.4 Conclusioni: il problema della rappresentazione	21
3.3.3 Problema dell'arresto (Halt problem) e sua indecidibilità	21

3.3.4	Problema indecidibile: equivalenza tra programmi	22
3.3.5	Modelli di Calcolo e complessità (tesi di Churc-Turing)	23
3.3.6	Algoritmi polinomiali ed esponenziali a confronto	23
3.4	Premessa alla teoria della complessità	24
3.5	Teoria della complessità	25
3.5.1	Tipologie di problemi	25
3.5.2	Ruolo di problemi decisionali e di ottimizzazione nella teoria	26
3.5.3	Teoria della verifica	26
3.5.3.1	Concetto di <i>certificato</i>	26
3.5.3.2	Verifica in tempo polinomiale	27
3.5.4	Classi di complessità	27
3.5.5	Riduzioni polinomiali	29
3.5.6	Problema NP-arduo	29
3.5.7	Problema NP-completo	29
3.5.8	Dimostrare che un problema è NP-arduo o NP-completo	30
3.5.9	Premessa al teorema di Cook: problema SAT	30
3.5.10	Teorema di Cook	31
3.5.11	Classi co-P e co-NP	31
3.5.12	Recap sulla gerarchia delle classi	32
II	Casualità e sequenze pseudocasuali	33
4	Casualità secondo Kolmogorov	34
4.1	Necessità di sequenze casuali	34
4.2	Idea generale	34
4.3	Complessità in un sistema di calcolo	35
4.3.1	Complessità di Kolmogorov	35
4.3.2	Sistema di calcolo universale	35
4.4	Definizione di casualità secondo Kolmogorov	36
4.5	Esistenza di sequenze casuali di qualunque lunghezza	36
4.6	Verifica della casualità di Kolmogorov problema indecidibile	37
5	Generatori pseudocasuali	38
5.1	Sorgente binaria casuale	38
5.2	Valutazione statistica	39
5.3	Generazione di sequenze brevi	40
5.3.1	Generatore di numeri pseudo-casuali	40
5.3.2	Esempio di generatore pseudo-casuale: generatore lineare	40
5.3.3	Generatore polinomiale, generalizzazione del precedente	41
5.3.4	Generatori crittograficamente sicuri con funzioni one-way	41
5.3.5	Generatori binari crittograficamente sicuri	42
5.3.6	Generatore BBS	42
5.3.7	Generatore di numeri pseudocasuali con cifrario simmetrico DES	43

6 Algoritmi randomizzati	44
6.1 Classificazione	44
6.2 Test di primalità (inefficiente)	44
6.3 Test di primalità: algoritmo di Miller-Rabin	45
6.3.1 Premessa	45
6.3.2 CN (ma non sufficienti) per il test di primalità	46
6.3.3 Lemma di Miller-Rabin (pochi composti verificano le CN)	46
6.3.4 Algoritmo completo e costo computazionale	47
6.4 Generazione di numeri primi	48
III Cifrari storici	49
7 Introduzione	50
7.1 Principi di Ruggero Bacone	50
8 Cifrari a sostituzione	51
8.1 Tipologie	51
8.2 Cifrario monoalfabetico: <i>cifrario di Cesare</i>	51
8.3 Cifrario monoalfabetico: <i>cifrario affine</i>	52
8.4 Cifrario monoalfabetico: <i>cifrario completo</i>	54
8.5 Cifrario polialfabetico: archivio fotografico di Augusto	55
8.6 Cifrario polialfabetico: disco dell'Alberti (XV secolo)	55
8.6.1 Primo metodo di cifratura	56
8.6.2 Secondo metodo di cifratura	56
8.7 Cifrario polialfabetico: De Vigenère	57
9 Cifrari a trasposizione	58
9.1 Cifrario a permutazione semplice	58
9.2 Cifrario a permutazione di colonne	59
9.3 Cifrario a griglia (<i>cifrario di Richelieu</i>)	59
10 Forzatura dei cfrari storici	61
10.1 Crittoanalisi statistica	61
10.2 Cifrari a sostituzione monoalfabetici	62
10.3 Cifrari a sostituzione polialfabetica	63
10.4 Cifrari a trasposizione	63
11 La macchina Enigma	64
11.1 Chiave e assetto iniziale	64
11.2 Interno della macchina	65

IV Cifrari perfetti	68
12 Introduzione	69
12.1 Formalizzazione del concetto di <i>cifrario perfetto</i>	69
12.1.1 Esempi di cifrari non perfetti (casi estremi)	70
12.2 Teorema di Shannon	70
13 One-Time Pad	72
13.1 Cifratura e decifratura	72
13.2 Teorema su <i>One-time pad</i> perfetto e minimale	73
13.3 Riduzione della dimensione della chiave	75
V Cifrari simmetrici moderni	76
14 Data Encryption Standard (DES)	77
14.1 Introduzione e principi di Shannon	77
14.2 Storia	77
14.3 Cifratura	78
14.4 Proprietà del DES con m, c complementati	82
14.5 Attacchi al DES	82
14.5.1 Attacchi esaurienti	83
14.5.2 Crittoanalisi differenziale	83
14.5.3 Crittoanalisi lineare	84
14.6 Varianti del DES	84
14.6.1 Scelta indipendente delle sottochiavi	84
14.6.2 Cifratura multipla: 2DES	84
14.6.3 Cifratura multipla: 3DES	85
15 Advanced Encryption Standard (AES)	86
15.1 Storia	86
15.2 Caratteristiche	87
15.2.1 Dimensione della chiave e numero di fasi	87
15.2.2 Preparazione: selezione della sottochiave di ogni fase	87
15.2.3 Preparazione: cifratura per blocchi	88
15.2.4 Fasi	88
15.2.4.1 <i>Substitute bytes</i> (S-box)	88
15.2.4.2 <i>Shift rows</i> (Diffusione)	89
15.2.4.3 <i>Mix columns</i> (Diffusione)	90
15.2.4.4 <i>Add round key</i> (Confusione)	90
15.2.5 <i>Cipher Block Chaining</i> (CBC)	90

VI Crittografia a chiave pubblica	92
16 Introduzione	93
16.1 Problema base: scambio delle chiavi su canali insicuri	93
16.2 Schema della crittografia a chiave pubblica	93
16.2.1 Caratteristiche del cifrario	94
16.2.2 Vantaggi e svantaggi	95
16.3 Generatori	95
16.3.1 Teorema sull'esistenza di almeno un generatore	96
16.3.2 Problema: individuare i generatori	96
16.3.2.1 Problema del logaritmo discreto	96
16.4 Funzioni <i>one-way trap-door</i>	97
16.4.1 Esempio: problema della fattorizzazione	97
16.4.2 Esempio: calcolo della radice in modulo	97
17 RSA (Rivest-Shamir-Adleman)	98
17.1 Generazione delle chiavi	98
17.2 Cifratura e decifratura	99
17.2.1 Esempio	99
17.3 Dimostrazione del teorema di correttezza del cifrario	100
17.3.1 Caso: il messaggio m è divisibile sia per q che per p	100
17.3.2 Caso: p e q non dividono il messaggio m	101
17.3.3 Caso: il messaggio m è divisibile per p e non per q (o viceversa)	101
17.4 Sicurezza del cifrario ed attacchi	102
17.4.1 Soluzioni equivalenti alla fattorizzazione	102
17.4.2 Osservazioni sulla fattorizzazione di n	103
17.4.3 Scelta ottimale dei valori p, q	103
17.4.4 Attacchi con esponenti (d ed e) bassi	104
17.4.5 Attacchi a tempo	104
17.4.6 Attacchi sulla scelta di e	105
17.4.7 Attacco con lo stesso valore di n (<i>common modulus</i>)	106
17.5 Cifrari ibridi: uso di RSA ed AES insieme	106
18 Protocollo Diffie-Hellman	107
18.1 Spiegazione	107
18.2 Sicurezza del problema del logaritmo discreto	108
18.2.1 Attacchi passivi	108
18.2.2 Attacchi attivi	108
19 Cifrario di El Gamal	110
19.1 Spiegazione	110
19.2 Dimostrazione di correttezza	111

20 Crittografia su curve ellittiche	112
20.1 Vantaggi della crittografia a curve ellittiche	112
20.2 Curve ellittiche	112
20.3 Curve ellittiche su $K = \mathbb{R}$	113
20.3.1 Punto opposto $-\mathbf{P}$	114
20.3.2 Somma sulla curva ellittica	114
20.3.2.1 Definizione di somma	115
20.3.2.2 Proprietà della somma	116
20.3.2.3 Formulazione algebrica	117
20.4 Curve ellittiche su campi finiti	117
20.4.1 Formulazione algebrica	118
20.4.2 Ordine di una curva ellittica (cardinalità)	119
20.4.3 Costruzione di una funzione <i>one-way trap-door</i>	120
20.4.3.1 Moltiplicazione scalare: algoritmo dei raddoppi ripetuti .	120
20.5 Protocollo Diffie-Hellman su curve ellittiche	121
20.6 Scambio di messaggi (equivalente di El-Gamal)	122
20.6.1 Algoritmo di Koblitz per la trasformazione del messaggio	122
20.6.2 Cifratura e decifrazione	122
20.6.3 Sicurezza	123
VII Applicazioni moderne della crittografia	124
21 Identificazione, autenticazione e firma digitale	125
21.1 Introduzione	125
21.1.1 Funzioni hash	126
21.1.1.1 Funzioni hash <i>one-way</i>	126
21.1.1.2 MD5 (Message Digest, v5)	127
21.1.1.3 RIPEMD-160	127
21.1.1.4 SHA (Secure Hash Algorithm)	127
21.2 Protocollo di identificazione su canali sicuri	128
21.3 Protocollo di identificazione su canali insicuri	129
21.4 Autenticazione su canale insicuro	130
21.4.1 Protocollo	130
21.4.2 <i>Message Authentication Code</i> (MAC)	130
21.5 Protocolli per la firma digitale	131
21.5.1 Caratteristiche della firma manuale e di quella digitale	131
21.5.2 Protocollo 1: messaggio in chiaro e firmato (DH)	131
21.5.3 Protocollo 2: messaggio cifrato e firmato	132
21.5.3.1 Schema applicato all'RSA	132
21.5.3.2 Attacco con giochi algebrici	133
21.5.4 Protocollo resistente agli attacchi	134
21.5.5 Attacchi man-in-the-middle	135
21.6 Certification Authority - CA	135
21.6.1 Contenuto del certificato digitale	135

21.7	Protocollo finale con certificazione digitale	136
22	Protocollo conoscenza zero (<i>Zero Knowledge</i>)	137
22.1	Idea Generale	137
22.1.1	Proprietà generali	138
22.2	Protocollo di identificazione Fiat-Shamir	138
22.2.1	Generazione della chiave da parte del Prover	139
22.2.2	Autenticazione	139
22.2.3	Dimostrazione di completezza e correttezza	140
22.2.4	Dimostrazione della proprietà Zero Knowledge	140
22.2.5	Sicurezza a confronto con altri protocolli	140
23	Protocollo SSL (<i>Secure Socket Layer</i>)	141
23.1	Introduzione	141
23.2	Livelli in cui SSL è organizzato	142
23.3	Fasi in SSL handshake	142
23.3.1	Messaggio <i>client hello</i> inviato dall'utente U	142
23.3.2	Messaggio <i>server hello</i> inviato dal server S	143
23.3.3	Autenticazione (creazione certificato da parte del sistema S)	143
23.3.4	Messaggio <i>server hello done</i>	143
23.3.5	Controllo del certificato da parte dell'utente U	143
23.3.6	Costruzione del <i>master secret</i> presso l'utente U	144
23.3.7	Ricostruzione del <i>master secret</i> presso il server S	144
23.3.7.1	Extra: Autenticazione dell'utente U (se richiesto certificato a U)	144
23.3.8	Messaggio <i>finished</i> inviato sia dall'utente che dal server	144
23.4	SSL Record	145
23.5	Sicurezza di SSL	145
24	Quantum Distribution Key (QDK)	148
24.1	Introduzione	148
24.1.1	Principi della meccanica quantistica	148
24.2	Protocollo BB84	149
24.2.1	Premessa: struttura generale dell'hardware	150
24.2.1.1	<i>Beam Splitter Polarizzante</i> (PBS)	150
24.2.2	Step del protocollo	151
24.2.3	Esempio	152
24.2.4	Presenza del crittoanalista ed errori sperimentali	152
25	Bitcoin	154
25.1	Introduzione	154
25.2	Funzionamento	155
25.2.1	Transazione	155
25.3	Frazionamento delle transazioni	156
25.3.1	Primo esempio con Bob	156

25.3.2 Secondo esempio con Alice	157
25.3.3 Validazione	157
25.4 Struttura di un blocco della <i>blockchain</i>	157
25.5 Miner e mining	158
VIII Appendici	159
A Nozioni di algebra modulare	160
A.1 Utilità dell'algebra modulare in crittografia	160
A.2 Proprietà principali dell'operatore modulo	160
A.3 Relazione di congruenza	161
A.3.1 Differenza tra relazioni di congruenza e di uguaglianza	161
A.4 Numeri coprimi (ribadiamo, da <i>Wikipedia</i>)	161
A.5 Insieme degli interi e insieme degli interi coprimi con n	161
A.6 Funzione di Eulero	162
A.7 Teorema di Eulero	162
A.7.1 Conseguenze dei risultati: formula per trovare l'inverso	163
A.8 Teorema sull'inverso: ammissione e numero di soluzioni	163
A.8.1 Corollario: unicità della soluzione	163
A.9 Recap delle nozioni affrontate fino ad ora	164
A.10 Algoritmo di Euclide esteso	164
A.10.1 Calcolo dell'inverso con l'algoritmo	165
A.11 Algoritmo delle quadrature successive (o esponenziazione veloce)	166
B Funzione <i>floor</i> e <i>ceil</i>	168
C Operatore XOR	169
D Esercizi (raccolta incompleta)	170

Capitolo 1

Alcuni dettagli sul corso

Obiettivo del corso e argomenti

L'obiettivo del corso è comprendere le nozioni moderne dei cifrari moderni.

- Faremo ciò da un punto di vista algoritmico, affrontando prima i cifrari storici . Sono richieste nozioni di algoritmica, ma anche di complessità computazionale.
- Vedremo che alla base di un qualunque meccanismo di cifratura serve una chiave, e per tale motivo entreremo nel concetto di casualità e di numeri casuali: la casualità è necessaria per avere buone chiavi.
- Vedremo cifrari perfetti, che ci proteggono in modo completo se usati in modo corretto (soluzione ideale, anche se con un costo molto elevato facendo riferimento alla generazione delle chiavi).
- Vedremo cifrari simmetrici, cifratura a chiave pubblica di prima generazione (RSA) e seconda generazione (crittografia su curve ellittiche).
- Affronteremo anche il protocollo di autenticazione e le firme digitali, soffermandoci brevemente sul protocollo SSL.
- Concluderemo con argomenti più avanzati:
 - protocollo “Zero Knowledge”;
 - elementi su valute virtuali e blockchain;
 - elementi di crittografia quantistica (protocollo per lo scambio delle chiavi, basato sui principi della meccanica quantistica).

Le nozioni relative all'ultimo punto non hanno nulla a che vedere col computer quantistico.

Libro suggerito Per seguire il corso è suggerito il libro

Bernasconi, Ferragina, Luccio, Elementi di crittografia, Pisa Università Press 2015

I diritti d'autore, molto bassi, sono devoluti all'associazione *Medici senza frontiere*.

Parte I

Introduzione

Capitolo 2

Introduzione alla crittologia

2.1 Crittografia e crittoanalisi

Il termine crittografia significa “scrittura nascosta”. Distinguiamo due tipologie di studi:

- **Crittografia** (metodi di cifratura).
Studio di tecniche matematiche per mascherare messaggi, rendendoli incomprensibili a chi non è il legittimo destinatario.
- **Crittoanalisi** (metodi di interpretazione).
Studio di tecniche matematiche per svelare messaggi criptati (si forza, si cerca di recuperare informazioni nascoste quando non si è il legittimo proprietario).

L'unione dei due termini da origine a un termine poco utilizzato: **crittologia**, lo studio della comunicazione su canali non sicuri e relativi problemi. La concezione del termine crittografia non è stretta: con questo termine facciamo molto spesso riferimento anche a questioni di crittoanalisi.

2.2 Scenario tipico della crittografia

Il tipico scenario in cui ci poniamo è quello in cui Alice e Bob vogliono comunicare un messaggio m su un canale insicuro dove è possibile intercettare i messaggi. Decidono quindi di adottare un **metodo di cifratura** che trasforma m in c (c è detto **crittogramma**), che deve essere:

- **incomprensibile** al crittoanalista (Eve - Eavesdropper d'ora in poi)
- **facilmente decifrabile** da Bob

2.2.1 Funzione di cifratura (criptaggio)

L'operazione con la quale si trasforma m in c è di fatto una funzione:

$$C : \text{msg} \longrightarrow \text{critto}$$

La funzione è applicata da chi spedisce il messaggio. Si passa dallo spazio dei messaggi allo spazio dei crittogrammi: chiaramente a un messaggio deve essere associato uno e un solo crittogramma (ricordare la definizione di funzione matematica¹).

2.2.2 Funzione di decifratura (decifrazione)

L'operazione inversa:

$$D : \text{critto} \longrightarrow \text{msg}$$

La funzione è applicata da chi riceve il messaggio e vuole decifrarlo. Si passa dallo spazio dei crittogrammi allo spazio dei messaggi.

2.2.3 Schema di comunicazione

$$\text{Alice} : m \xrightarrow{C} c \xrightarrow[\text{canale insicuro}]{} c \xrightarrow{D} m : \text{Bob}$$

Si noti che per funzionare C e D devono essere in tempo polinomiale mentre per il crittoanalista, noto c , deve essere esponenziale il tempo utile per riottenere m .

NB C e D devono essere l'una l'inversa dell'altra:

$$D(c) = D(C(m)) = m$$

quindi C è iniettiva: m diversi vanno in c diversi (altrimenti il destinatario Bob non potrà determinare in modo univoco il messaggio a lui rivolto).

2.3 Esempi antichi

Erodoto in "Storie" (V secolo a.C.) Si prende un servitore, si rasano i suoi capelli e si scrive il messaggio sulla sua testa, si aspetta che la ricresca lo copra e poi si spedisce il servitore verso Bob che dovrà solamente rasarla nuovamente.

Spartani (V secolo a.C.) Gli spartani (V secolo a.C.) usavano lo scitale che è un'asta cilindrica costruita in due esemplari identici posseduti dai due corrispondenti. Su un pezzo di pelle viene scritto il messaggio dopo averlo avvolto attorno al cilindro seguendo le linee di esso. La fettuccia viene poi fatta indossare da un'uomo che la porta al ricevente.

Enea Tattico (Grecia, IV Secolo a.C.) Enea tattico dedica un intero capitolo ai metodi militari usati per scambiarsi i messaggi:

- inviare un libro con alcune lettere sottolineate a formare il messaggio in chiaro
- sostituire le vocali con altri simboli

¹Da Wikipedia. In matematica, una funzione è una relazione tra due insiemi, chiamati dominio e codominio della funzione, che associa a ogni elemento del dominio uno e un solo elemento del codominio.

Cifrario di Cesare Il Cifrario di Cesare è il più antico cifrario di concezione moderna. c è ottenuto da m sostituendo ogni lettera con quella a 3 posizioni più avanti:

$$\begin{aligned} A &\longrightarrow D \\ B &\longrightarrow E \\ C &\longrightarrow F \\ D &\longrightarrow G \\ \dots &\longrightarrow \dots \end{aligned}$$

Nella decifrazione faremo la stessa cosa al contrario. La segretezza in questo caso dipende dalla conoscenza del metodo: se il metodo viene scoperto allora non può essere più utilizzato!

2.4 Livello di segretezza

I metodi crittografici si classificano in:

- **per uso ristretto** (usati in ambito diplomatico e/o militare²): in cui la parte segreta del meccanismo è ampia (C e D sono tenute segrete)
- **per uso generale** (rivolto alla crittografia di massa): in cui la parte segreta è molto limitata (si restringe alla sola *chiave*, nota solo ai due endpoint della comunicazione)

NB: per i cifrari di massa quindi le regole sono pubbliche, solo le chiavi sono segrete. **Occorre sempre pensare che il nemico conosca il sistema.**

Proporzione Maggiore è la dimensione della parte segreta del cifrario minore deve essere il numero delle persone a conoscenza della parte segreta.

2.4.1 Ridefinizione delle funzioni di criptaggio e decifrazione

Ridefiniamo quindi:

$$c = C(m, k) \quad m = D(c, k)$$

con k chiave segreta diversa per ogni coppia di utenti. Se non si conosce k la conoscenza dell'algoritmo non deve permettere l'estrazione di informazioni dal crittogramma. Se una chiave viene divulgata basta generarne un'altra lasciando inalterati C e D . Ovviamente l'insieme delle chiavi deve essere così grande da non essere rompibile tramite brute-force e deve essere scelta in modo casuale.

NB: brute-force \equiv *attacco esauriente*

²Si tenga conto che maggiore è il numero di persone coinvolte, maggiore è il rischio che il segreto venga rivelato da persone che ne sono a conoscenza.

Esempio Se $|key| = 10^{20}$ ed un calcolatore impiegasse 10^{-6} secondi per calcolare $D(c, k)$ e verificarne la significatività occorrerebbero comunque milioni di anni per provarle tutte. NB: solo la grandezza dello spazio delle chiavi non è un buon indice per l'affidabilità di un cifrario, potrebbe sempre essere rotto matematicamente (la segretezza non deve dipendere esclusivamente dalla cardinalità dello spazio).

2.5 Crittoanalisi

Il crittoanalista può assumere uno dei seguenti comportamenti.

- Comportamento *passivo*: si limita ad ascoltare il canale
- Comportamento *attivo*: disturba le comunicazioni o modifica il contenuto dei messaggi

Gli attacchi hanno lo scopo di forzare il sistema, raggiungendo l'informazione protetta da quel particolare sistema. L'attacco al sistema crittografico può avere due esiti:

- **successo**, si scopre la funzione D, o si trova la chiave (e quindi possiamo decifrare qualunque crittogramma);
- **successo limitato**, si scoprano informazioni parziali, ma sufficienti per comprendere il contenuto del messaggio.

2.5.1 Tipologie di attacchi

Gli attacchi dipendono dalle informazioni in possesso del crittoanalista:

- **cipher text attack** (solo testo cifrato): si hanno una serie di crittogrammi:

$$c_1, \dots, c_r$$

- **known plain-text attack** (testo in chiaro noto): il crittoanalista ha a disposizione delle coppie

$$(m_1, c_1), \dots, (m_r, c_r)$$

i messaggi non sono scelti dal crittoanalista

- **chosen plain-text attack** (testo in chiaro scelto): ci si procura una serie di coppie

$$(m_1, c_1), \dots, (m_r, c_r)$$

relative a messaggi in chiaro scelti dal crittoanalista

2.5.2 Attacchi man-in-the-middle

Il crittoanalista si installa sul canale ed interrompe le comunicazioni dirette tra i due, le sostituisce con messaggi propri e convince ogni utente che quei messaggi provengono legittimamente dall'altro.

Il crittoanalista si finge Alice agli occhi di Bob e viceversa.

Capitolo 3

Rimandi alla teoria dell'informazione

3.1 Rappresentazione matematica di oggetti

Per parlare di Crittografia dobbiamo introdurre alcune nozioni matematiche relative alla rappresentazione di oggetti. Rappresentare matematicamente un oggetto significa attribuirgli un nome: dobbiamo fare ciò in modo rigoroso.

Alfabeto Un *alfabeto* è un insieme finito di caratteri detti simboli.

3.1.1 Rappresentazione degli oggetti con l'alfabeto

Un oggetto è rappresentato da una sequenza ordinata di caratteri dell'alfabeto. A oggetti diversi corrispondono sequenze diverse ed il numero di oggetti rappresentabili è infinito (fissata una lunghezza qualsiasi posso sempre creare sequenze più lunghe). Fissato un alfabeto Γ tale che $|\Gamma| = s$ (dove s è il numero di caratteri dell'alfabeto) e fissati N oggetti da rappresentare otteniamo:

- $d(s, N)$: è la lunghezza della sequenza più lunga di un oggetto dell'insieme
- $d_{min}(s, N)$: valore minimo di $d(s, N)$ tra tutte le rappresentazioni possibili

Un metodo di rappresentazione è tanto migliore tanto più $d(s, N)$ si avvicina a $d_{min}(s, N)$.

3.1.1.1 Alfabeto unario

Prendiamo un alfabeto con un solo simbolo ($s = 1$): $\Gamma = \{0\}$

Variazione di N L'unica possibilità che abbiamo per costruire sequenze diverse è aumentare la lunghezza N , visto che le sequenze sono solo ripetizioni dell'unico carattere presente nell'alfabeto (0, 00, 000, ...).

$$\implies d_{min}(1, N) = N$$

La rappresentazione non è comoda, ma soprattutto è poco efficiente.

3.1.1.2 Alfabeto binario

Passiamo a un alfabeto con due caratteri ($s = 2$).

$$\Gamma = \{0, 1\}$$

$\forall k \geq 1$ (dove k è la lunghezza della sequenza) si hanno 2^k sequenze, quindi il numero totale di sequenze lunghe da 1 a k è (data la lunghezza k consideriamo anche le possibili sequenze realizzabili con un numero di caratteri minori a k):

$$\sum_{i=0}^k 2^i = 2^{k+1} - 2$$

Nel risultato finale si ha -2 invece di -1 visto che non si considera $i = 0$. Per N oggetti da rappresentare imponiamo che il numero di sequenze possibili sia minimo uguale ad N :

$$\begin{aligned} 2^{k+1} - 2 &\geq N \\ 2^{k+1} &\geq N + 2 \\ (k+1)\log(2) &\geq \log(N+2) \\ (k+1) &\geq \frac{\log(N+2)}{\log(2)} \\ k &\geq \log_2(N+2) - 1 \end{aligned}$$

troviamo il minimo applicando l'operatore *ceil*

$$\begin{aligned} k_{min} &= d_{min}(2, N) = \lceil \log_2(N+2) - 1 \rceil \\ \lceil \log_2 N \rceil - 1 &\leq d_{min}(2, N) \leq \lceil \log_2 N \rceil \end{aligned}$$

Esempio Facciamo un esempio con $N = 7$ (vogliamo sette sequenze di caratteri diverse), abbiamo $\lceil \log_2 7 \rceil = 3$, dunque possiamo costruire N sequenze differenti tutte di lunghezza $\lceil \log_2 7 \rceil$

000, 001, 010, 011, 100, 101, 110

La cosa è importante nelle applicazioni crittografiche: parole di pari lunghezza per evitare l'uso di separatori (cioè simboli che mi segnalano la fine di una parola e l'inizio della successiva).

3.1.1.3 Alfabeto s -ario

Abbiamo un alfabeto caratterizzato da s caratteri. Possiamo costruire N sequenze differenti tutte di $\lceil \log_s N \rceil$ caratteri.

Esempio Supponiamo di avere Γ e di voler generare 1000 sequenze:

$$\Gamma = \{0, \dots, 9\} \implies \lceil \log_{10} 1000 \rceil = 3 \text{ cioè i numeri da 000 a 999}$$

Usare sequenze tutte della stessa lunghezza è vantaggioso perché posso concatenare più oggetti senza usare un separatore. Si dice **rappresentazione efficiente** una rappresentazione che usa un numero massimo di caratteri di ordine logaritmico nella cardinalità dell'insieme da rappresentare (N) quindi bisogna avere almeno 2 caratteri.

3.1.2 Rappresentazione di interi

La rappresentazione degli interi è una rappresentazione di tipo posizionale, dove il valore del numero dipende anche dalla posizione. Risulta efficiente indipendentemente dalla base $s \geq 2$ scelta perché un intero N costituito da d cifre soddisfa la seguente condizione:

$$\lceil \log_2 N \rceil \leq d \leq \lceil \log_2 N \rceil + 1$$

Abbiamo il concetto di **riduzione logaritmica** tra il valore di un numero N e la lunghezza della sua rappresentazione. Per avere complessità polinomiale il numero di istruzioni dell'algoritmo deve crescere come il logaritmo del valore (e non come il valore!!!).

3.2 Differenza tra calcolabilità e complessità

A questo punto possiamo distinguere la teoria della calcolabilità da quella della complessità:

- **Teoria della calcolabilità**

La teoria della calcolabilità studia gli algoritmi da un punto di vista di risoluzione algoritmica: quali problemi possono essere risolti algoritmicamente in un calcolatore (problemI decidibili) e quali no (problemI non decidibili).

- **Teoria della complessità**

La teoria della complessità studia gli algoritmi (problemI decidibili) da un punto di vista dell'efficienza (problema efficiente, complessità polinomiale, o problema intrattabile, complessità esponenziale)

3.3 Teoria della calcolabilità

La calcolabilità si occupa della potenza e dei limiti dei sistemi di calcolo. La disciplina ha un secolo di vita, nasce quando i logici matematici iniziano ad esplorare i concetti di computazione, algoritmo e risoluzione algoritmica di problemI.

3.3.1 Problemi computazionali

I problemI a cui siamo interessati sono problemI computazionali, che hanno una formulazione matematica precisa e una risoluzione algoritmica. Si distinguono in:

- **problemI non decidibili**, non risolvibili algoritmicamente in un calcolatore (ne è stata dimostrata l'esistenza prima della nascita del calcolatore);
- **problemI decidibili**, che ammettono una soluzione algoritmica. Se consideriamo la complessità distinguiamo questi problemI in
 - **problemI trattabili** (algoritmi di costo polinomiale, esistono algoritmi di complessità polinomiale che ne permettono la risoluzione),

- **problemi intrattabili** (algoritmi di costo esponenziale, non solo non esistono algoritmi di complessità polinomiale, ma si è anche dimostrata l'esistenza di un limite inferiore),
- **problemi aventi stato computazionale non noto** (non esistono algoritmi di complessità polinomiale, ma non si è dimostrata l'esistenza di un limite inferiore - dunque potrebbero essere scoperti in futuro algoritmi aventi minore complessità).

3.3.2 Esistenza di problemi non decidibili

Vogliamo dimostrare l'esistenza di problemi non decidibili.

3.3.2.1 Definizione: insiemi con stessa cardinalità

Due insiemi A e B hanno la stessa cardinalità

$$|A| = |B|$$

se e solo se si può stabilire una corrispondenza biunivoca tra i loro elementi (una mappa)¹.

3.3.2.2 Definizione: insieme numerabile

Un insieme è *numerabile* (ha una infinità numerabile di elementi) *se e solo se* i suoi elementi si possono mettere in corrispondenza biunivoca con i numeri naturali.

- Un insieme numerabile può essere elencato:

$$a_1, a_2, \dots, a_n, \dots$$

- Le stringhe di lunghezza finita di simboli di un alfabeto finito sono un insieme numerabile

- Il numero di queste stringhe è infinito, ma la cosa importante è che **ogni sequenza dell'elenco abbia una distanza finita dall'inizio** (cioè deve essere possibile raggiungere qualsiasi sequenza attraverso un numero finito di passi dalla prima sequenza).

- Usiamo l'*ordinamento canonico*:

- * si ordinano le sequenze per lunghezza crescente
- * le sequenze di pari lunghezza si ordinano alfabeticamente (supponendo di avere creato una regola di ordine tra i caratteri)

Quindi una sequenza s arbitraria si troverà tra quelle $|s|$, posizione alfabetica tra queste.

¹Rinfreschiamoci la memoria sulla biunivocità grazie a Wikipedia: in matematica una corrispondenza biunivoca tra due insiemi X e Y è una relazione binaria tra X e Y , tale che ad ogni elemento di X corrisponda uno ed un solo elemento di Y , e viceversa ad ogni elemento di Y corrisponda uno ed un solo elemento di X . In particolare, la corrispondenza biunivoca è una relazione di equivalenza.

– **Esempio.**

$$\begin{aligned}\Gamma &= \{a, b, c, \dots, z\} \\ a, b, c, \dots, z \\ aa, ab, \dots, az, ba, \dots, bz, \dots, za, \dots, zz \\ aaa, aab, \dots, baa, \dots, zaa, \dots, zzz \\ aaaa, \dots\end{aligned}$$

Seguendo questo metodo ogni sequenza corrisponde ad un numero $\in \mathbb{N}$ ed ogni naturale ha una sequenza associata.

- **Attenzione.** Questo si può fare perché abbiamo preso sequenze di lunghezza finita, per sequenze di lunghezza infinita non esiste una enumerazione.

• **Esempi di insiemi non numerabili.**

- \mathbb{R} e sottoinsiemi (per esempio \mathbb{R} ristretto a $(0, 1)$ o $[0, 1]$)
- insieme delle funzioni in una o più variabili

3.3.2.3 Dimostrazione

Dimostriamo per quanto riguarda l'ultimo insieme citato tra i non numerabili.

- Un problema computazionale può essere visto come una funzione matematica che associa ad ogni insieme di dati su k numeri interi il risultato su j numeri interi:

$$f : \mathbb{N}^k \longrightarrow \mathbb{N}^j$$

Se siamo in grado di dimostrare che questo insieme di funzioni non è numerabile allora possiamo dire la stessa per l'insieme di problemi computazionali.

- Per semplificarci la vita dimostriamo la cosa su un sottoinsieme

$$F = \{f | f : \mathbb{N} \longrightarrow \{0, 1\}\}$$

Ogni $f \in F$ può essere rappresentata da una sequenza infinita:

$$\begin{array}{c|c|c|c|c|c|c|c|c} x & | & 0 & | & 1 & | & 2 & | & 3 & | & - & | & n & | & - \\ \hline f(x) & | & 0 & | & 1 & | & 0 & | & 1 & | & - & | & 0 & | & - \end{array}$$

oppure da una regola finita di costruzione:

$$f(x) = \begin{cases} 0 & \text{se } x \text{ è pari} \\ 1 & \text{se } x \text{ è dispari} \end{cases}$$

- Supponiamo per assurdo che F sia numerabile, quindi è possibile trovare una enumerazione per $f \in F$: questo significa poter associare un numero progressivo nella numerazione e costruire una tabella avente un numero infinito di righe (possibili funzioni) e colonne (possibili valori in ingresso).

x	0	1	2	-
$f_0(x)$	1	0	1	-
$f_1(x)$	0	0	1	-
$f_2(x)$	1	1	0	-

- Per dimostrare la non numerabilità dell'insieme mi basta individuare una sola funzione non numerabile. Consideriamo dunque la seguente funzione:

$$g(x) = \begin{cases} 0 & \text{se } f_x(x) = 1 \\ 1 & \text{se } f_x(x) = 0 \end{cases}$$

$g \in F$ perché è una funzione dai naturali a $\{0, 1\}$ ma non può corrispondere a nessuna delle funzioni nella tabella precedente:

- non può essere f_0 in quanto differisce in $x = 0$
- non può essere f_1 in quanto differisce in $x = 1$
- e così via $\forall n$

- Diciamo la cosa in modo più formale:

- Supponiamo per assurdo che esista un valore i tale che $f_i \equiv g$.
- Possiamo dire che $f_i \equiv g$, tuttavia per definizione

$$g(j) = \begin{cases} 0 & f_j(j) = 1 \\ 1 & f_j(j) = 0 \end{cases}$$

Quanto scritto consiste di fatto nella funzione di complementazione, quindi per forza ho $g(j) \neq f_j(j)$. Noi abbiamo considerato la diagonale, ma nulla ci vieta di considerare linee arbitrarie che attraversano la tabella toccando tutte le righe e tutte le colonne esattamente una volta.

- Per qualunque numerazione scelta esiste sempre almeno una funzione esclusa, dunque F non è numerabile.
- Se F è un insieme non numerabile allora tutti gli insiemi che lo contengono non sono numerabili:

- * $f : \mathbb{N} \rightarrow \mathbb{N}$
- * $f : \mathbb{N} \rightarrow \mathbb{R}$
- * $f : \mathbb{R} \rightarrow \mathbb{R}$
- * $f : \mathbb{N}^k \rightarrow \mathbb{N}^j$ (insieme delle funzioni in una o più variabili)

L'insieme dei problemi computazionali non è numerabile!

3.3.2.4 Conclusioni: il problema della rappresentazione

L'informatica rappresenta tutte le sue entità in forma digitale come "sequenze finite di simboli di alfabeti finiti". Possiamo dire quindi che la conoscenza umana è numerabile. L'algoritmo è una sequenza finita di operazioni, completamente e univocamente determinate. La formulazione di un algoritmo dipende dal modello di calcolo utilizzato: se uso la macchina di Turing ho una codifica, se uso il C ne ho un'altra, ecc. Qualsiasi codifica si scelga gli algoritmi sono sempre descritti da una sequenza finita di caratteri di un alfabeto finito.

In conclusione

- L'insieme degli algoritmi è infinito ma numerabile.
- I problemi computazionali sono infiniti ma non numerabili.

Se l'insieme dei problemi computazionali non è numerabile allora non è possibile stabilire una corrispondenza biunivoca tra insieme dei problemi computazionali e quello degli algoritmi, ergo **esistono problemi computazionali non associati a un corrispondente algoritmo di calcolo** (ricordare quanto detto all'inizio su due insiemi che hanno la stessa cardinalità). ■

3.3.3 Problema dell'arresto (Halt problem) e sua indecidibilità

Il problema dell'arresto è un problema indecidibile, individuato da Alan Turing nel 1930.

- Presi ad arbitrio un algoritmo A e i suoi dati di input d, dobbiamo decidere in tempo finito (volendo anche esponenziale, basta che sia finito) se la computazione di A su d termina (il programma termina la sua esecuzione) o non termina (va in loop, cioè continua a ripetere la stessa sequenza di istruzioni all'infinito).
- In sostanza vogliamo individuare **un algoritmo che indaga sulle proprietà di un altro algoritmo**, cioè un algoritmo che ha come input un altro algoritmo.
 - Un algoritmo A, comunque formulato, può operare sulla rappresentazione di un altro algoritmo B. Poniamo quindi A(B).
 - Possiamo fare anche A(A), per esempio se l'algoritmo A deve calcolare la lunghezza del file che rappresenta l'algoritmo.

Dimostrazione di Turing sull'indecidibilità del problema

- Anche questa volta ragioniamo per assurdo: supponiamo che esista un algoritmo ARRESTO che prende A e d come input, e determina la risposta in tempo finito. In particolare

$$\text{ARRESTO}(A, d) = \begin{cases} 1 & A(d) \text{ termina} \\ 0 & A(d) \text{ non termina} \end{cases}$$

Risulta scontato che l'algoritmo non deve consistere nell'eseguire il programma: questo perché se il problema non si arresta allora la funzione non può restituire 0 in tempo finito.

- Immaginiamo $D = A$, quindi $A(A)$ (algoritmo che ha come input se stesso). Possiamo dire

$$\text{ARRESTO}(A, A) = 1 \implies A(A) \text{ termina}$$

fin qui nulla di strano.

- Se esiste l'algoritmo ARRESTO allora esiste il seguente algoritmo PARADOSSO:

`PARADOSSO(A) : while(ARRESTO(A, A)) { }`

la condizione nel while è true e non viene influenzata dal corpo (non cambia).

- L'algoritmo $\text{PARADOSSO}(A)$ non termina se l'algoritmo $A(A)$ termina, cioè se abbiamo $\text{ARRESTO}(A, A) = 1$. Posso dire che $\text{PARADOSSO}(A)$ termina solo se $A(A)$ non termina.

$$\text{PARADOSSO}(A) \text{ termina} \iff x = \text{ARRESTO}(A, A) = 0$$

- Consideriamo $\text{PARADOSSO}(\text{PARADOSSO})$: osservo che termina solo se

$$\text{PARADOSSO}(\text{PARADOSSO}) \text{ termina} \iff x = \text{ARRESTO}(\text{PARADOSSO}, \text{PARADOSSO}) = 0$$

Ecco la contraddizione: stiamo dicendo che $\text{PARADOSSO}(\text{PARADOSSO})$ termina solo se $\text{PARADOSSO}(\text{PARADOSSO})$ non termina. La contraddizione può essere risolta solo affermando la non esistenza di PARADOSSO, dunque non può esistere nemmeno ARRESTO. ■

Un algoritmo ARRESTO sarebbe uno strumento estremamente potente: permetterebbe, ad esempio, di dimostrare congetture ancora aperte sugli interi (esempio: la congettura di Goldbach²).

3.3.4 Problema indecidibile: equivalenza tra programmi

E' indecidibile stabilire l'equivalente tra due programmi (stesso input \implies stesso output)

Turing afferma che non esistono algoritmi che decidono il comportamento di altri algoritmi esaminandoli dall'esterno, cioè senza passare dalla loro simulazione.

²Da Wikipedia. In matematica, la congettura di Goldbach è uno dei più vecchi problemi irrisolti nella teoria dei numeri. Essa afferma che ogni numero pari maggiore di 2 può essere scritto come somma di due numeri primi (che possono essere anche uguali).

3.3.5 Modelli di Calcolo e complessità (tesi di Churc-Turing)

Poniamoci una domanda

La decidibilità dipende dal modello di calcolo o è una caratteristica del problema?

La domanda può essere posta in altri modi:

- i linguaggi di programmazione esistenti sono tutti equivalenti?
- esistono linguaggi più potenti e/o più semplici di altri?
- ci sono algoritmi descrivibili in un linguaggio, ma non in un altro?
- è possibile che problemi attualmente irrisolvibili possano essere risolti in futuro con altri linguaggi o con altri calcolatori?

Non abbiamo un teorema che ci permetta di rispondere in modo univoco alle domande precedenti, ma abbiamo una tesi secondo cui il modello di calcolo non influisce sulla decidibilità (tesi di Churc-Turing): **tutti i ragionevoli modelli di calcolo risolvono esattamente la stessa classe di problemi.**

Significato Questo significa che i modelli di calcolo si equivalgono nella possibilità di risolvere problemi (anche se con diversa efficienza), quindi incrementi qualitativi alla struttura di una macchina (anche al computer quantistico), o alle istruzioni di un linguaggio di programmazione, servono solo a:

- abbassare il tempo di esecuzione;
- rendere più agevole la programmazione.

3.3.6 Algoritmi polinomiali ed esponenziali a confronto

Perché si pone questa demarcazione netta tra algoritmi di costi polinomiale e algoritmi di costo esponenziale? Supponiamo di avere un calcolatore C_1 e un calcolatore C_2 , entrambi hanno a disposizione un tempo t e devono risolvere lo stesso problema (per cui abbiamo lo stesso algoritmo risolutivo). Vogliamo vedere come cambia la dimensione dei dati trattabili a parità di tempo, in macchine con capacità computazionali diverse. Quindi:

- n_1 sono i dati trattabili nel tempo t su C_1 , n_2 sono i dati trattabili nel tempo t su C_2
- Usare C_2 per un tempo t equivale a usare C_1 per un tempo $k \cdot t$

Algoritmo polinomiale	Algoritmo esponenziale
L'algoritmo risolve un problema in $c \cdot n^s$ secondi (c è la costante che solitamente ignoriamo nella complessità).	L'algoritmo risolve un problema in $c \cdot 2^n$ secondi (c è la costante che solitamente ignoriamo nella complessità).
$C_1: c \cdot n_1^s = t \rightarrow n_1 = (t/c)^{1/s}$ $C_2: c \cdot n_2^s = k \cdot t \rightarrow n_2 = (k \cdot t/c)^{1/s} = k^{1/s}(t/c)^{1/s}$ Morale della favola: $n_2 = k^{1/s}n_1$ Miglioro di un fattore moltiplicativo dipendente da k .	$C_1: c \cdot 2^{n_1} = t \rightarrow 2^{n_1} = t/c$ $C_2: c \cdot 2^{n_2} = k \cdot t \rightarrow 2^{n_2} = kt/c = k \cdot 2^{n_1}$ Morale della favola: $n_2 = n_1 + \log_2 k$ Miglioro di un fattore additivo dipendente da k .

Si osserva che il miglioramento dell'algoritmo esponenziale è minore rispetto al miglioramento dell'algoritmo polinomiale. L'algoritmo efficiente è molto più importante della piattaforma hardware usata:

Un algoritmo inefficiente lo è su qualunque piattaforma.

3.4 Premessa alla teoria della complessità

Per capire quanto è efficiente un algoritmo è necessario chiarire i parametri su cui valutare l'efficienza. Solitamente si prende a riferimento il tempo di esecuzione oppure l'occupazione di memoria. In entrambi i casi si descrive l'andamento attraverso una funzione matematica.

- Successivamente dobbiamo determinare la variabile indipendente su cui si lavora: solitamente si prende il parametro n , che è la dimensione dell'ingresso (o dimensione dei dati). Il parametro indica la lunghezza della sequenza che specifica il valore dei dati per una particolare istanza del problema.
- Consideriamo la seguente funzione per la moltiplicazione

Molt(a,b)

Supponiamo di avere i numeri $a = 351, b = 66$. Possiamo porre $n = 5$ poichè le cifre tra i due numeri sono cinque. A questo punto sorge il problema che il numero di cifre dipende dalla base di rappresentazione adottata. Come risolviamo? Ripensiamo a quanto visto per la rappresentazione di oggetti attraverso sequenze finite costituite da caratteri di un alfabeto finito: per costruire N sequenze differenti ho bisogno del seguente numero di caratteri

$$\lceil \log_n N \rceil$$

dove n è il numero di caratteri dell'alfabeto. Il numero di caratteri dell'alfabeto è letteralmente la base (la base n è costituita da n cifre che vanno da 0 ad $n - 1$). Sappiamo che è possibile fare un cambio di base nel logaritmo con la seguente formula

$$\log_c a = \frac{\log_b a}{\log_b c} \longrightarrow (\log_c a) (\log_b c) = \log_b a$$

Questo significa che nel passaggio da una base a un'altra si va a moltiplicare per un fattore costante. Si consideri per esempio un intero: esso è rappresentato in base 10 col seguente numero di cifre

$$\lceil \log_{10} N \rceil$$

Per passare dalla base 10 alla base 2 andrà a moltiplicare il numero di cifre per $\log_2 10$. Da questo deduciamo la necessità di ragionare per **ordini di grandezza**.

- L'ordine di grandezza è un insieme infinito che comprende tutte le funzioni che hanno un particolare comportamento rispetto a una funzione $g(n)$. Esse sono:

- $O(g(n))$, l'insieme di tutte le funzioni $f(n)$ per cui esistono due costanti positive c, n_0 tali che

$$f(n) \leq cg(n)$$

questo $\forall n \geq n_0$. Al crescere di n il comportamento della funzione $f(n)$ è limitato superiormente dalla funzione $g(n)$

- $\Omega(g(n))$, l'insieme di tutte le funzioni $f(n)$ per cui esistono due costanti positive c, n_0 tali che

$$f(n) \geq cg(n)$$

questo $\forall n \geq n_0$. Al crescere di n il comportamento della funzione $f(n)$ è limitato inferiormente dalla funzione $g(n)$

- $\Theta(n)$, l'insieme di tutte le funzioni $f(n)$ per cui esistono due costanti positive c, n_0 tali che

$$c_1g(n) \leq f(n) \leq c_2g(n)$$

questo $\forall n \geq n_0$. Al crescere di n il comportamento della funzione $f(n)$ è limitato superiormente e inferiormente dalla funzione $g(n)$. Segue

$$f(n) = \Theta(g(n)) \implies f(n) = O(g(n)) \text{ e } f(n) = \Omega(g(n))$$

3.5 Teoria della complessità

Possiamo immaginare un problema Π come una funzione matematica $f : I \rightarrow S$, dove:

- I : insieme delle *istanze* di ingresso
- S : insieme delle *soluzioni*

3.5.1 Tipologie di problemi

Alcune tipologie di problemi sono:

- **Problemi decisionali**

Problemi dove l'insieme delle soluzioni è un insieme binario $S = \{0, 1\}$.

- Un'istanza $x \in I$ è detta istanza positiva o accettabile se $\Pi(x) = 1$

- Un’istanza $x \in I$ è detta istanza negativa se: $\Pi(x) = 0$
- **Esempi:** numero primo, grafo connesso...

- **Problemi di ricerca**

Data un’istanza $x \in I$ vogliamo trovare una soluzione s (trovare un cammino tra due vertici, ecc)

- **Problemi di ottimizzazione**

Data un’istanza $x \in I$ si vuole trovare la migliore soluzione s tra tutte quelle ammissibili (cammino minimo, ecc)

3.5.2 Ruolo di problemi decisionali e di ottimizzazione nella teoria

La teoria della complessità fa riferimento alla sola classe decisionale in quanto:

- essendo $s = \{0, 1\}$ non ci si deve preoccupare del tempo per restituire la soluzione (influisce solo il tempo per il calcolo della soluzione);
- la difficoltà è già presente nella sua versione decisionale.

Molti problemi di interesse pratico sono problemi di ottimizzazione. Possiamo esprimere i problemi di ottimizzazione in forma decisionale.

Esempio

- MAX-CLIQUE: trovare la clique più grande in un grafo G
- CLIQUE decisionale: esiste una clique in G di almeno k vertici?

CLIQUE decisionale non è più difficile di MAX-CLIQUE. Supponiamo di saper risolvere MAX-CLIQUE: se io so trovare la risposta allora automaticamente trovo risposta alla versione decisionale del problema. Il problema di ottimizzazione è almeno tanto difficile quanto il corrispondente problema decisionale. Lo studio del problema decisionale permette di fornire un limite inferiore alla versione di ottimizzazione.

3.5.3 Teoria della verifica

3.5.3.1 Concetto di *certificato*

In un problema decisionale siamo interessati a verificare se un’istanza del problema soddisfa una certa proprietà. In alcuni problemi è possibile fornire un certificato per ogni istanza accettabile, che ci convinca della sua accettabilità. Questo significa che non siamo tanto interessati alla soluzione nello specifico, ma a un qualcosa che attesti la sua esistenza.

Definizione Si definisce il **certificato** come un attestato ”breve” di esistenza di una soluzione con determinate proprietà (una descrizione succinta della soluzione), si definisce solo per le istanze accettabili (gli attestati di non esistenza non sono facili da costruire).

3.5.3.2 Verifica in tempo polinomiale

Nella teoria della verifica utilizziamo il costo della verifica di un certificato per una istanza positiva per caratterizzare la complessità del problema stesso.

Definizione Un problema Π è **verificabile in tempo polinomiale** se

- ogni istanza accettabile x di Π di lunghezza n ammette un certificato y di lunghezza polinomiale in n
- esiste un algoritmo di verifica polinomiale in n ed applicabile ad ogni coppia $\langle x, y \rangle$ che attesta se x è accettabile.

Utilità della teoria della verifica ($\Pi \in NP?$) Quanto è effettivamente utile la teoria della verifica? Il dubbio può venire visto che il certificato è un qualcosa di molto vicino alla soluzione. La teoria della verifica ci aiuta a costruire una gerarchia di complessità dei problemi (cioè capire quanto un problema è facile o difficile), ma non aiuta nella risoluzione dei problemi stessi.

- Chi ha una soluzione può verificare in tempo polinomiale se l'istanza è veramente accettabile (classe NP).
- Chi non ha una soluzione non ha il certificato e deve individuarlo. Lo faccio con metodo forza bruta: in tempo esponenziale considero tutti i casi possibili.

Abbiamo introdotto la teoria della verifica per affrontare, poco più avanti, le classi NP ed NP-completo.

3.5.4 Classi di complessità

Dato Π decisionale ed A algoritmo diciamo che A risolve Π se, dato l'input x :

$$A(x) = 1 \iff \Pi(x) = 1$$

Diciamo poi che A risolve Π in tempo $t(n)$ ed in spazio $s(n)$ se il tempo di esecuzione e l'occupazione di memoria di A hanno andamento simile alle funzioni $t(n)$ e $s(n)$. Dato $f(n)$ diremo:

- **Time($f(n)$)**
Insieme dei problemi decisionali che possono essere risolti in tempo $O(f(n))$
- **Space($f(n)$)**
Insieme dei problemi decisionali che possono essere risolti in spazio $O(f(n))$
- **Classe P.**
E' la classe di problemi che possono essere risolti in AL PIU' tempo polinomiale nella dimensione dell'istanza di input.

- **Algoritmo polinomiale (tempo).** Date due costanti $c, n_0 > 0$ tale che il numero di passi elementari è al più n^c per ogni input di dimensione n e per ogni $n > n_0$.

- **Classe PSPACE.**

Equivalenti della classe P nello spazio. E' la classe di problemi che possono essere risolti in spazio polinomiale nella dimensione dell'istanza di input.

- **Classe RP.**

La classe di problemi *Random-Polinomial* contiene tutti i problemi che ammettono un algoritmo di costo polinomiale randomizzato. Il test di primalità appartiene a questa classe.

$$P \subseteq RP \subseteq NP$$

- **Classe EXP-TIME**

La classe $\text{Exp}(\text{time})$ è la classe dei problemi risolvibili in **AL PIU'** tempo esponenziale nella dimensione n dell'istanza di input.

$$P \subseteq PSpace \subseteq \text{Exp-Time}$$

NB: $P \subseteq PSpace$ perché un algoritmo polinomiale può accedere al più ad un numero polinomiale di locazioni di memoria (altrimenti dovrebbe essere esponenziale).

NB: $PSpace \subseteq \text{Exp-Time}$ (abbiamo detto **AL PIU'**, nota bene)

Ad oggi non sappiamo se sono inclusioni proprie, l'unica separazione dimostrata è

$$P \subset \text{Exp-Time}$$

poiché abbiamo problemi risolti in *Exp* e non in *P* (ad esempio Hanoi)

- **Classe NP**

NP è la classe dei problemi decisionali verificabili in tempo polinomiale.

NB: *NP* sta per *polinomiale su macchine non deterministiche*

Abbiamo quindi che se si ha una soluzione essa è facile da verificare ma se non si ha una soluzione la si cerca in tempo esponenziale.

NB: $P \subseteq NP$ certamente perché qualsiasi problema in *P* può essere risolto in tempo polinomiale. Non sappiamo però se $P \subset NP$ o $P = NP$. Per il momento la congettura pone $P \neq NP$



Si prenda ad esempio la SAT (lo vediamo nel dettaglio poco più avanti): è facile capire se dei valori soddisfano una formula logica, ma non è altrettanto facile individuare dei valori che soddisfano la formula logica stessa (forza brutta, analizzo tutte le possibili combinazioni).

- **Problemi NP-completi**

I ricercatori, per cercare di rispondere al dilemma di P ed NP, hanno individuato all'interno della classe NP i problemi più difficili: la classe NP-completo.

Questi problemi hanno una caratteristica comune: se esistesse un algoritmo polinomiale per risolvere uno solo di questi problemi, allora tutti i problemi in NP potrebbero essere risolti in tempo polinomiale.

La definizione è posta poco più avanti, a seguito della riduzione del concetto di *riduzione polinomiale*.

3.5.5 Riduzioni polinomiali

Presi i problemi decisionali Π_1 e Π_2 , con insiemi di input I_1 e I_2 (rispettivamente di Π_1 e di Π_2), diremo che il problema Π_1 si riduce in tempo polinomiale al problema Π_2 se esiste una funzione che mi trasforma una istanza del primo problema in una istanza del secondo problema

$$f : I_1 \rightarrow I_2$$

Essa è calcolabile in tempo polinomiale. Rappresentiamo la cosa con la seguente notazione:

$$\Pi_1 \leq_p \Pi_2$$

inoltre

$$x \text{ è accettabile per } \Pi_1 \iff f(x) \text{ è accettabile per } \Pi_2, \forall x \in \Pi$$

Utilità La riduzione polinomiale è utile perché supponendo di risolvere Π_2 in tempo polinomiale allora Π_1 viene tradotto in tempo polinomiale in Π_2 e quindi anche Π_1 è polinomiale:

$$\Pi_1 \leq_p \Pi_2 \text{ e } \Pi_2 \in P \implies \Pi_1 \in P$$

3.5.6 Problema NP-arduo

Un problema (non per forza decisionale) Π si dice NP-arduo se ogni problema $\Pi' \in NP$ può essere risolto utilizzando Π (riducendo polynomialmente il problema Π' al problema Π):

$$\forall \Pi' \in NP \quad \Pi' \leq_p \Pi$$

3.5.7 Problema NP-completo

Un problema decisionale Π si dice NP-Completo se

- $\Pi \in NP$

- Π è NP-arduo

Dimostriamo che se troviamo Π NP-Completo, ma $\Pi \in P$ allora $P = NP$:

- per ogni $\Pi' \in NP$ si ha $\Pi' \leq_p \Pi$, quindi trasformo $I_{\Pi'}$ in I_Π .
- I_Π è risolubile in tempo polinomiale, quindi qualunque $\Pi' \in NP$ è risolto in tempo polinomiale.

■

3.5.8 Dimostrare che un problema è NP-arduo o NP-completo

Abbiamo visto che dimostrare $\Pi \in NP$ è semplice: basta esibire un certificato polinomiale (teoria della verifica). Non è semplice invece dimostrare che un problema è NP-arduo o NP-completo perché:

- devo dimostrare che tutti i problemi NP si riducono a Π
- ma la prima definizione di NP-completo aggira il problema: il *Teorema di Cook*

3.5.9 Premessa al teorema di Cook: problema SAT

Sia dato un insieme V di variabili logiche, definiamo:

- **letterale**: una variabile o una sua negazione
- **clausola**: disgiunzione (OR) di letterali
- **espressione booleana in forma normale congiuntiva** (FNC): è una espressione logica formata da clausole unite da congiunzioni (AND di OR di variabili dirette o negate)

Es: dati $V = \{x, y, z, w\}$ una possibile FNC potrebbe essere:

$$(x \vee \bar{y} \vee z) \wedge (\bar{x} \vee w) \wedge y$$

SAT si occupa di cercare dei valori di verità per rendere vera l'espressione. Nell'esempio la FNC è soddisfatta per

$$x = 1 \quad y = 1 \quad z = 0 \quad w = 1$$

NB Il problema c'è solo se l'espressione è FNC e passare ad un'altra forma richiede tempo esponenziale.

Risoluzione Per risolvere iteriamo sulle 2^n possibili combinazioni e controlliamo la soddisfattibilità.

$$\text{SAT} \in \text{Exp-Time}$$

quindi per risolvere questo ed altri problemi (clique, cammino hamiltoniano) è necessario iterare tra tutte le possibili combinazioni di ingresso? *Non lo sappiamo.*

3.5.10 Teorema di Cook

Cook ha dimostrato che il problema SAT è NP-Completo. Dati un qualunque problema NP ed una qualunque istanza x si può dimostrare che una espressione booleana in forma normale congiuntiva che descrive l'algoritmo del problema è sempre costruibile.

Conseguenza Qualsiasi problema NP si riduce a SAT. Per dimostrarlo quindi che un problema è NP-completo ci basta prenderne uno che lo è e provare a ridurlo al problema in studio.

Esempio Per dimostrare che clique è NP-completo cerchiamo un algoritmo polinomiale tale che:

$$\text{SAT} \leq_p \text{CLIQUE}$$

se lo troviamo allora CLIQUE è NP-completo. SAT e CLIQUE sono NP equivalenti: *tutti i problemi NP completi sono tra di loro NP equivalenti*.

3.5.11 Classi co-P e co-NP

C'è una profonda differenza tra certificare l'esistenza di una soluzione e certificarne la non esistenza. Dato un problema decisionale Π possiamo definire un problema decisionale $co - \Pi$ che accetta tutte e solo le istanze rifiutate dal problema Π .

- **Insieme co-P.**

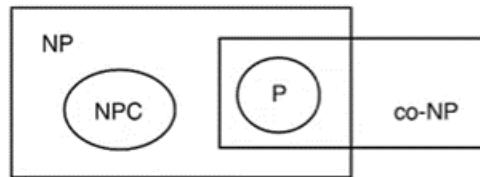
Il problema $co - \Pi$ appartiene alla classe co-P. Si osservi che per ogni problema $co - \Pi$ appartenente alla classe co-P si ha $co - \Pi \in P$. Se il problema Π è polinomiale allora anche il problema $co - \Pi$: risolvo il primo e complemento ottenendo il secondo (la cosa vale in entrambi i sensi).

- **Insieme co-NP.**

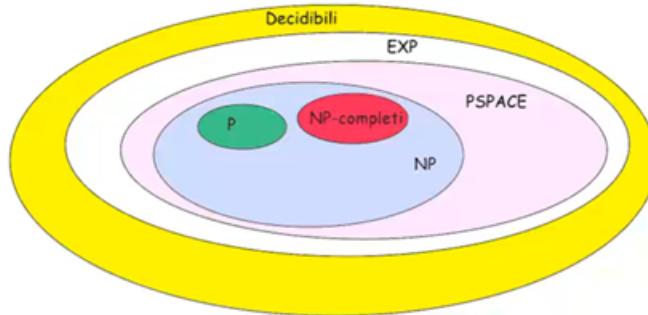
Come abbiamo l'insieme co-P per P abbiamo anche l'insieme co-NP per NP. Si osservi che per ogni problema decisionale $co - \Pi$ appartenente alla classe co-NP si ha $co - \Pi \in NP$. Contrariamente a prima si congettura che le due classi siano diverse (se ho il certificato di un problema non posso costruire direttamente il certificato del problema complementato): è solo una congettura, se questa fosse vera allora si potrebbe confermare che $P \neq NP$ (P è uguale al suo complementare, se dimostro che P è in realtà diverso si dimostra la disegualanza).

$$P \neq co-NP$$

Esempio: trovare ciclo hamiltoniano (NP) e trovare ciclo non-hamiltoniano (co-NP).



3.5.12 Recap sulla gerarchia delle classi



- La gerarchia si costruisce usando ciò che sappiamo con certezza (P contenuto in PSPACE) e le varie congetture.
- L'insieme più vasto è quello dei problemi decidibili.
- Abbiamo EXPTIME, all'interno del quale è presente PSPACE.
- All'interno di NP abbiamo P ed NP-completo. Un esempio di problema che non è P, ma neanche NP-completo, è la fattorizzazione (problema che ci interessa per la Crittografia). Questa cosa non è a caso: sulla macchina quantistica il problema è facilmente risolvibile.

Inoltre

- La classe P contiene problemi risolvibili in tempo polinomiale.
- La classe NP contiene i problemi verificabili in tempo polinomiale (remember, verificare significa ottenere un certificato che attesta l'esistenza di una soluzione rispetto a una particolare istanza I)
- I problemi NP-hard sono problemi a cui è possibile applicare il concetto di riduzione polinomiale: tutti i problemi appartenenti alla classe NP possono essere ridotti al problema NP-hard.
- La classe NP-completo include tutti i problemi che appartengono ad NP e sono NP-hard. Se io risolvo uno di questi problemi risolvo tutti i problemi. Per verificare la possibilità di ridurre tutti i problemi di NP a un particolare problema uso il teorema di Cook, che afferma che qualsiasi problema NP si riduce a SAT: quindi se io posso ridurre SAT al problema in studio allora tutti i problemi NP possono essere ridotti al problema in studio.
- Dato un problema posso definire il complementare, che accetta tutte le istanze rifiutate dal problema iniziale. Questo secondo problema appartiene alla classe co-P, ma anche a P. I problemi che appartengono ad NP appartengono anche a co-NP: si congettura che $P \neq \text{co-NP}$ (dato il certificato del problema iniziale non riesco a costruire automaticamente il certificato del problema complementato).

Parte II

Casualità e sequenze pseudocasuali

Capitolo 4

Casualità secondo Kolmogorov

4.1 Necessità di sequenze casuali

Data una sequenza binaria vogliamo capire se questa è casuale oppure no. Ne abbiamo bisogno per vari motivi:

- la generazione della chiave (deve essere imprevedibile);
- algoritmi randomizzati in Crittografia (algoritmi alimentati da sequenze casuali).

4.2 Idea generale

Poniamoci quindi un quesito: data una sequenza si vuole sapere se essa sia stata generata casualmente o meno. Consideriamo due sequenze binarie:

$$\bullet \quad h = 111111111111\dots 1 \qquad \bullet \quad h' = 101101101011\dots 0$$

La prima sequenza può facilmente essere descritta come *n volte 1*, mentre la seconda può essere descritta dettandola. La probabilità di generare ognuna le due sequenze è

$$P(h) = P(h') = \left(\frac{1}{2}\right)^n = \frac{1}{2^n}$$

questo perchè la probabilità che esca 0, ma anche 1, in una cifra è 1/2. Supponiamo di creare un algoritmo A_h per generare la prima sequenza:

$$A_h : < \text{scrivi } n \text{ volte } 1 >$$

L'unica cosa che varia è la lunghezza della sequenza h , quindi

$$|h| = n \qquad |A_h| = \log_2 |h| + c = \log_2 n + c = O(\log n)$$

Lavoreremo con la seguente intuizione: *una sequenza è casuale se non ammette un algoritmo di generazione la cui rappresentazione binaria sia più corta di h*. In parole povere: il modo più economico per definire una sequenza casuale è assegnare la sequenza stessa.

4.3 Complessità in un sistema di calcolo

I sistemi di calcolo (modello di calcolo astratto come una RAM, macchina di Turing, circuito di calcolo... , il luogo dove girano i nostri algoritmi) costituiscono un insieme infinito e numerabile: $S_1, S_2, \dots, S_i, \dots$. Dobbiamo scegliere un sistema di calcolo per mostrare che la casualità è indipendente. Sceglieremo S_i e supponiamo di avere un programma p che genera una sequenza h nel sistema di calcolo scelto.

4.3.1 Complessità di Kolmogorov

Definiamo la complessità di Kolmogorov di h nel sistema S_i come:

$$K_{S_i}(h) = \min\{|p| : S_i(p) = h\}$$

cioè la lunghezza minima di un programma p che nel sistema di calcolo i -esimo genera la sequenza h .

Sequenza irregolare Se la sequenza è irregolare il programma dovrà contenerla per intero, quindi anche il programma più piccolo che la genera sarà più lungo di h :

$$K_{S_i}(h) = |h| + c_i$$

dove c_i è la parte di programma che trasferisce in output la sequenza (dipende dal sistema di calcolo i -esimo scelto).

4.3.2 Sistema di calcolo universale

Per svincolarci dal sistema di calcolo consideriamo un sistema di calcolo universale S_u che è in grado di simulare tutti i sistemi di calcolo esistenti:

$$S_i(p) = h \iff S_u(< i, p >) = S_i(p) = h$$

Si ha quindi $q = < i, p >$, programma che genera la sequenza h nel sistema universale S_u . Esso è lungo:

$$|q| = |i| + |p| = \lceil \log_2 i \rceil + |p|$$

Il primo termine dipende dall'indice del sistema e non dalla sequenza h (è palesemente una costante, adottato un sistema i). E' logico derivare:

$$\forall h, \forall i : K_{S_u}(h) \leq K_{S_i}(h) + c_i$$

dove c_i dipende dal sistema di riferimento. La complessità del sistema universale per generare una sequenza h è minore o uguale rispetto alla somma tra una costante e la complessità del sistema i -esimo per generare la solita sequenza.

- L'uguaglianza si ha per quelle sequenze h generate per simulazione di S_i , nel caso in cui non esistano algoritmi più brevi per S_u

- Il minore vale per le sequenze generabili con un programma più corto (per esempio simulando un altro sistema di calcolo $j \neq i$).

Dimentichiamoci per un attimo della costante: possiamo vedere che la complessità di Kolmogorov di una sequenza in un qualunque sistema di calcolo non potrà mai essere minore della complessità nel sistema di calcolo universale.

$$K_{S_i}(h) \geq K_{S_u}(h)$$

Otteniamo un limite inferiore.

Quindi Definiamo la complessità di Kolmogorov di una sequenza h è quella nel suo sistema universale:

$$K(h) = K_{S_u}(h)$$

Eliminiamo il pedice: non facciamo riferimento a un particolare sistema di calcolo.

4.4 Definizione di casualità secondo Kolmogorov

Una sequenza è casuale secondo Kolmogorov se:

$$K(h) \geq |h| - \lceil \log_2 |h| \rceil$$

Si conferma che la casualità è una proprietà della sequenza, indipendente dal sistema di calcolo adottato (e quindi dal modo in cui viene generata la sequenza).

4.5 Esistenza di sequenze casuali di qualunque lunghezza

Fissato la lunghezza n abbiamo $S = 2^n$ sequenze aventi tale lunghezza. Poniamo T come numero di sequenze NON casuali di lunghezza n . Si vuole dimostrare che

$$T < S$$

cioè affermare che per ogni lunghezza n esistono sequenze casuali.

Dimostrazione Definiamo N come numero di sequenze di lunghezza $< n - \lceil \log_2 n \rceil$ (sequenze che non rispettano la definizione di Kolmogorov), sono esattamente:

$$N = \sum_{i=1}^{n-\lceil \log_2 n \rceil} 2^i = 2^{n-\lceil \log_2 n \rceil} - 1$$

Tra queste N sequenze ci sono necessariamente anche le sequenze che descrivono i programmi p per generare tutte le T sequenze non casuali, quindi necessariamente si ha $T \leq N$. Chiaramente abbiamo $N < S$ e quindi

$$T \geq N < S \longrightarrow T < S$$

Osservazione Al crescere di n le sequenze casuali sono molto maggiori delle sequenze non casuali:

$$\frac{T}{S} \leq \frac{N}{S} = \frac{2^{n-\lceil \log_2 n \rceil} - 1}{2^n} = \frac{1}{2^{\lceil \log n \rceil}} - \frac{1}{2^n} < \frac{1}{2^{\lceil \log n \rceil}}$$

calcolando il limite concludiamo:

$$\lim_{n \rightarrow \infty} \frac{T}{S} = 0 \implies T \ll S$$

Non solo T è minore di S , ma le sequenze casuali per Kolmogorov sono maggioritarie. ■

4.6 Verifica della casualità di Kolmogorov problema indecidibile

Purtroppo verificare la casualità di Kolmogorov è un problema indecidibile.

Dimostrazione Supponiamo per assurdo che esista un algoritmo:

$$\text{RANDOM}(h) = \begin{cases} 1 & \text{se } h \text{ è casuale} \\ 0 & \text{altrimenti} \end{cases}$$

Costruiamo l'algoritmo PARADOSSO che non ha parametri di input ed enumera tutte le possibili sequenze binarie in ordine di lunghezza crescente.

PARADOSSO:

```
for binary h <- 1 to inf do{
    if( |h| - ceil(log2(|h|)) > |p| && RANDOM(h) == 1 )
        return h
}
```

Usiamo dal for non appena individuiamo una sequenza casuale e di dimensione maggiore della lunghezza $|p|$ definita come

$$|p| = |\text{PARADOSSO}| + |\text{RANDOM}|$$

Si noti che $|p|$ è costante in quanto n viene preso come parametro e non è presente all'interno del programma. Dato che esistono sequenze casuali di qualunque lunghezza questo programma si fermerà sicuramente, fornendoci la prima sequenza casuale che soddisfa il vincolo di dimensione. Attenzione all'assurdo:

- si richiede che il programma sia di lunghezza breve e generi h ;
- si richiede che la sequenza h sia casuale.

Con un programma di lunghezza breve non è possibile avere una sequenza h casuale secondo Kolmogorov. Ne segue quindi che RANDOM non può esistere! ■

Capitolo 5

Generatori pseudocasuali

Abbiamo visto che esistono sequenze casuali per qualunque lunghezza arbitraria (secondo la definizione di Kolmogorov), ma che la verifica della casualità è un problema indecidibile. Oggi introduciamo **sequenze pseudocasuali**, facendo esempi pratici.

5.1 Sorgente binaria casuale

Una sorgente binaria casuale genera una sequenza di bit con le seguenti caratteristiche:

- **Pari probabilità.**

$$P(0) = P(1) = \frac{1}{2}$$

vedremo che questa proprietà non è così vitale, possiamo indebolirla ponendo $P(0) > 0, P(1) > 0$ e immutabili durante il processo di generazione.

- **Generazione di un bit indipendente.**

La generazione di un bit è indipendente dalla generazione degli altri. Non posso prevedere il valore di un bit a partire da quelli già generati.

Prendiamo un caso dove $P(0) > P(1)$. Chiaramente la sequenza conterrà più 0 che 1. Supponiamo di avere la seguente sequenza

00|11|00|11|10|00|01|01|00

Possiamo bilanciare la sequenza prendendo i bit a coppie. Elimino le sottosequenze uguali ed associo ad ogni coppia un valore 0 o 1:

$$01 \implies 0 \qquad \qquad \qquad 10 \implies 1$$

Otteniamo:

100

Questa sequenza ottenuta sarà la nostra sequenza casuale.

Esistono vere sorgenti casuali? Non è possibile garantire la perfetta casualità di una sorgente, e soprattutto l'indipendenza (generazione di bit indipendente da quella delle altre): ogni esperimento modifica l'ambiente, e quindi può influenzare gli esperimenti successivi. Dobbiamo accettare dei compromessi.

5.2 Valutazione statistica

Come possiamo valutare se il nostro generatore è valido da un punto di vista statistico? Vogliamo valutare le sequenze prodotte da un generatore pseudocasuale, cioè valutare se una certa sequenza presenta le proprietà tipiche di una sequenza casuale. Per simulazioni e algoritmi randomizzati sono necessari i seguenti test:

- **Test di frequenza**

Verifica che le cifre siano presenti con la stessa frequenza

- **Poker test**

Verifica che le sotto sequenze di lunghezza fissa siano distribuite in modo equo.

- **Test di autocorrelazione**

Verifica che non ci siano regolarità nella sequenza ottenuta.

- **Run test**

Verifica che le sottosequenze massimali di elementi tutti ripetuti abbiano una distribuzione esponenziale negativa, cioè all'aumentare della loro lunghezza la frequenza decresce.

Per le applicazioni crittografiche si richiede anche il **test di prossimo bit**: test molto severo che implica tutti gli altri (nel senso che se si supera questo test gli altri sono sicuramente superati). Dobbiamo verificare l'impossibilità di fare previsioni sui bit prima che questi vengano generati.

Cosa si intende con test di prossimo bit? Deve essere impossibile fare previsioni con risorse di calcolo polinomiali. Un generatore binario supera il test di prossimo bit se non esiste un algoritmo polinomiale in grado di prevedere l' $i+1$ -esimo bit della sequenza a partire dalla conoscenza degli i bit precedentemente generati con probabilità maggiore di $\frac{1}{2}$.

Algoritmo crittograficamente sicuro Si dice che un generatore è crittograficamente sicuro se supera il test di prossimo bit. Il generatore lineare non supera questo test.

5.3 Generazione di sequenze brevi

Vediamo alcune tecniche di generazione di sequenze brevi:

- Fenomeni casuali presenti in natura (sorgenti di casualità tipo rumore del microfono)
- Processi software (come ad esempio la posizione della testina dell'hard disk o l'orologio del computer)
- Generazione mediante algoritmi matematici. Questo ultimo tipo costituisce i **generatori di numeri pseudo-casuali**. Non sono ovviamente sequenze casuali secondo Kolmogorov in quanto prodotte da un programma breve

Ci interessano questi ultimi generatori.

5.3.1 Generatore di numeri pseudo-casuali

Perchè pseudo? Tipicamente questi algoritmi sono brevi e deterministici, quindi le sequenze ottenute non sono per nulla casuali secondo la teoria di Kolmogorov.

Input Si parte dal seed, che è un piccolo valore casuale che l'utente può generare ricorrendo a sorgenti casuali, oppure lanciando una moneta. La casualità del seme viene usata per ottenere una sequenza più lunga.

Output Flusso di bit arbitrariamente lungo, periodico (sottosequenza periodo al suo interno, che si ripete). Maggiore è la lunghezza del periodo, migliore è il generatore.

Numero di sequenze generabili

- Sia S il numero di bit del seme.
- Sia n la lunghezza della sequenza ottenuta col generatore.

Tipicamente $n \gg S$. Si capisce che ci sono dei limiti: se io uso sempre lo stesso seme ho un algoritmo deterministico, viene generata sempre la solita sequenza “casuale”. Il numero di sequenze diverse ottenibili dal seme è 2^S , ma il numero di sequenze possibili con n bit è 2^n , $2^n \gg 2^S$

5.3.2 Esempio di generatore pseudo-casuale: generatore lineare

L'algoritmo non è valido per applicazioni crittografiche: i primi quattro test vengono superati, mentre il test di prossimo bit no. Questo perchè esistono algoritmi che osservando alcuni valori riescono a risalire ai parametri del generatore in tempo polinomiale.

Definizione Il generatore lineare è definito dalla seguente equazione. Si consideri la i -esima cifra:

$$x_i = (a \cdot x_{i-1} + b) \bmod m$$

a, b, m sono interi positivi. Il seme è il valore iniziale x_0 (che supponiamo di aver estratto casualmente).

Periodo e permutazione Data l'operazione modulo vogliamo ottenere una permutazione degli elementi appartenenti a $\mathbb{Z}_m = \{0, 1, \dots, m-1\}$: per fare ciò dobbiamo massimizzare il periodo, e massimizzare il periodo significa ottenere un periodo lungo m . Ci servono:

- b ed m coprimi, $\text{MCD}(b, m) = 1$;
- $(a - 1)$ deve essere divisibile per ogni fattore primo di m ;
- $(a - 1)$ deve essere un multiplo di 4 se m lo è.

Esempio Valori $a = 7, b = 7, m = 9$, seme $x_0 = 3$

$$\begin{aligned} x_i &= (7x_{i-1} + 7) \bmod 9 \\ &\implies 3, 1, 5, 6, 4, 8, \dots, 3 \end{aligned}$$

5.3.3 Generatore polinomiale, generalizzazione del precedente

Vediamo la versione generalizzata del generatore lineare, dove si pone un polinomio. Anch'essa supera i primi quattro test ma non quello del prossimo bit. E' nella forma:

$$x_i = (a_1 x_{i-1}^t + a_2 x_{i-1}^{t-1} + \dots + a_t x_{i-1} + a_{t+1}) \bmod n$$

Possiamo utilizzarlo come generatore di sequenze binarie, dividendo per n :

$$r = \frac{x_i}{n} \longrightarrow \begin{cases} \text{Se la prima cifra decimale di } r \text{ è pari} \implies 0 \\ \text{Se la prima cifra decimale di } r \text{ è dispari} \implies 1 \end{cases}$$

5.3.4 Generatori crittograficamente sicuri con funzioni one-way

Una funzione *one-way* è una funzione facile da calcolare (tempo polinomiale) ma difficile da invertire (tempo esponenziale):

$$\begin{aligned} x &\xrightarrow{f(x)} y : \text{tempo polinomiale} \\ y &\xrightarrow{f^{-1}(y)} x : \text{tempo esponenziale} \end{aligned}$$

Data una funzione f one-way e dato un seme x_0 calcolo una sequenza:

$$S: \left| \begin{array}{cccccc} x & f(x) & f(x_1) = f(f(x)) & \dots & f^{(n)}(x) = f(x_{n-1}) \\ x_0 & x_1 & x_2 & & x_n \end{array} \right.$$

si prende il seme e si itera la funzione *one-way* un numero arbitrario di volte. Il fatto è che ogni elemento della sequenza S si può calcolare polynomialmente dal precedente, ma non dai valori successivi (perché appunto la funzione è one-way).

Problema Se conosco x la sequenza è prevedibile! Risolviamo *restituendo la sequenza al contrario!* Dato x_{i+1} non possiamo calcolare facilmente x_i ma solo x_{i+2} . Bisognerebbe passare per una funzione esponenziale ad ogni passo.

5.3.5 Generatori binari crittograficamente sicuri

Come faccio a trasformare un generatore crittograficamente sicuro definito sugli interi in un generatore crittograficamente sicuro che produce una sequenza binaria? Tipicamente le funzioni one-way non sono funzioni 0 e 1, quello che si fa è estrarre la complessità computazionale della funzione one-way sintetizzandola in un unico bit. Usiamo i cosiddetti *predicati hard-core* delle funzioni one-way.

Cosa intendiamo con predicato in generale? *Il predicato è una funzione che restituisce un valore vero o falso*

Definizione di predicato hard-core $b(x)$ si dice predicato hard-core di una funzione one-way se:

- $b(x)$ è facile da calcolare conoscendo x
- $b(x)$ è difficile da prevedere conoscendo solo $f(x)$

Esempio Consideriamo la funzione *one-way*

$$x \rightarrow f(x) = x^2 \bmod n \text{ (n non primo)}$$

Consideriamo il seguente predicato *hard-core*

$$b(x) = \text{"x è dispari"}$$

Se prendiamo i valori $n = 77, x = 10$ diventa facile calcolare

$$f(x) = 10^2 \bmod 77 = 23$$

il contrario (conosciamo solo $f(x)$) prevede una enumerazione di tutti i valori da 0 a $n - 1$. Vediamo un esempio concreto col prossimo generatore.

5.3.6 Generatore BBS

Il generatore BBS (Blum-Blum-Shub), nato nel 1986 è un generatore crittograficamente sicuro! Sia $n = p \cdot q$ un prodotto tra numeri primi dove:

- $p \bmod 4 = 3$
- $q \bmod 4 = 3$
- $\text{MCD}\left(2\lfloor\frac{p}{4}\rfloor + 1, 2\lfloor\frac{q}{4}\rfloor + 1\right) = 1$

Per prima cosa si calcola il seme x_0 :

$$x_0 = y^2 \bmod n$$

successivamente si calcola una successione di $m \leq n$ interi con:

$$x_i = (x_{i-1})^2 \bmod n$$

In parallelo a ciò calcoliamo i vari predicati

$$\begin{aligned} b_0 &= 1 \iff x_n \text{ è dispari} \\ b_1 &= 1 \iff x_{n-1} \text{ è dispari} \\ &\dots \\ b_n &= 1 \iff x_0 \text{ è dispari} \end{aligned}$$

Otteniamo la seguente sequenza: $b_0 b_1 \dots b_n$

Difetti del generatore I problemi di questo generatore sono: la lentezza, la necessità di numeri molto grandi e l'esecuzione di potenze di questi numeri già molto grandi. Ci sono altre tecniche usate, più veloci anche se meno sicure.

5.3.7 Generatore di numeri pseudocasuali con cifrario simmetrico DES

Utilizziamo un cifrario simmetrico e una chiave. Prima di costruire il crittogramma sostituisco il messaggio con un *seme*. I cifrari simmetrici lavorano a blocchi, nel senso che producono parole di una certa lunghezza di bit: prendo un blocco, cifro e ottengo un output di un certo numero di bit. Vediamo un esempio che sfrutta il DES (funzione di cifratura C) ed è stato approvato dall'agenzia federale statunitense FIPS (*Federal Information Processing Standard*):

Generatore(s, m):

```
d = <rappresentazione su r bit di data ed ora attuali>
y = C(d, k)
z = s
for(i = 1; i <= m; i++) {
    xi = C(y XOR z, k);
    z = C(y XOR xi, k);
    <comunicazione x_i all'esterno>
}
```

L'input è costituito dal seme s (di r bit, in rDES solitamente 64 bit) e il numero di parole da produrre m (con parole intendiamo il numero di blocchi del DES). k è la chiave segreta del cifrario. Quello che facciamo è...

- Generare un valore y sfruttando la chiave segreta del cifrario e una rappresentazione di data e ora attuali. Questo valore viene utilizzato in tutte le iterazioni.
- Pongo come z iniziale il seme s posto in ingresso.
- Applico le funzioni indicate nello pseudocodice, con tanto di operatore XOR.

Se la cosa suona strana tornarci dopo aver fatto il DES.

Capitolo 6

Algoritmi randomizzati

6.1 Classificazione

Gli algoritmi randomizzati sono algoritmi dove eventi casuali influiscono sulla struttura dell'algoritmo stesso: oltre ai dati in ingresso influiscono valori casuali che ne determinano l'evoluzione. Li classifichiamo in due tipi:

- **Las Vegas**

Algoritmi con risultato **sicuramente** corretto in un tempo probabilmente breve

- **Monte Carlo**

Algoritmi con risultato **probabilmente** corretto in un tempo sicuramente breve (test di primalità di Miller-Rabin, potrei avere risultati non corretti). Questi algoritmi offrono tuttavia la possibilità di scegliere l'errore con il quale si ottiene il risultato. La probabilità di errore deve essere:

- arbitrariamente piccola, e matematicamente misurabile.

Questo ci permetterà di rendere l'errore trascurabile.

6.2 Test di primalità (inefficiente)

Consideriamo il seguente test di primalità e valutiamone l'efficienza

Primo (\sqrt{n}) → **INTERO DI CUI VOLGO TESTARE LA PRIMALITÀ**

for ($i = 2; i \leq \sqrt{n}, i++$)
 if ($N \% i == 0$) **return** **false**; ← **HO TROVATO UN DIVISORE**
 return **true**; ← **NON HO TROVATO DIVISORE**

SE N NON È PRIMO AVRA ALMENO UN DIVISORE

FACCIA MO LA VALUTAZIONE DI COMPLESSITÀ

$I = N \leftarrow$ L'ISTANZA DI INPUT È IL NUMERO N

$|I| = O(\log N) = m \leftarrow$ NUM. DI CIFRE CHE MI SERVONO PER SCRIVERE N

NEI CASO PEGGIORE IL NUMERO DI ITERAZIONI È \sqrt{N}

IL COSTO DEL CORPO DEL FOR È LA DIVISIONE, QUINDI $O(\log^2 N)$

$$T(m) = O(\sqrt{N} \cdot \log^2 N) = O(2^{m/2} \cdot m^2)$$

$2^m = N \text{ DA } |I|$

ALGORITMO PSEUDOPOLINOMIALE

DIMENSIONE

POLINOMIALE CON N (VALORE)

ESPOENZIALE CON m (DIMENSIONE)

NELL'ALGORITMO ABBIANO L'ENUMERAZIONE DI TUTTE LE SEQUENZE BINARIE DI $\frac{m}{2}$ BIT.

L'algoritmo è pseudopolinomiale, cioè:

- polinomiale rispetto al valore;
- esponenziale rispetto alla dimensione del valore (non tengo conto solo del numero, ma anche di quanto è grande il numero).

Vogliamo un algoritmo più efficiente: introduciamo per questo il test di primalità secondo Miller-Rabin, basato su sequenze casuali.

6.3 Test di primalità: algoritmo di Miller-Rabin

6.3.1 Premessa

Vogliamo testare un intero dispari N su n bit. Se N è dispari allora $N - 1$ è pari.

$$N - 1 = 2^\omega z$$

con z dispari e 2^ω la più grande potenza di 2 che divide $N - 1$. Vediamo due esempi:

$$N = 17 \implies N - 1 = 16 = 2^4 \cdot 1 \implies z = 1, \omega = 4$$

$$N = 21 \implies N - 1 = 20 = 2^2 \cdot 5 \implies z = 5, \omega = 2$$

z e ω ci serviranno nell'algoritmo: si noti che entrambi possono essere trovati in tempo polinomiale in quanto posso dividere un numero per 2 al massimo $\log_2 N$ volte quindi abbiamo un algoritmo polinomiale nel numero di cifre di N ($n = \log_2 N$)

6.3.2 CN (ma non sufficienti) per il test di primalità

Supponiamo che N sia un numero primo. Si prenda y , un intero casuale arbitrario tale che $2 \leq y \leq N - 1$ (che prende il nome di *testimone*). Affermiamo che per N valgono le seguenti proposizioni:

$$P1 : \text{MCD}(N, y) = 1$$

$$P2 : (y^z \bmod N = 1) \text{ OR } \left(\exists i : 0 \leq i \leq \omega - 1 \text{ t.c. } y^{2^i \cdot z} \bmod N = -1 \right)$$

Se il numero è primo allora vale sia P1 che P2. In P2 basta che una delle condizioni sia valida. Possiamo usare la veridicità di entrambi i predicati come condizione necessaria per il nostro test di primalità, tuttavia non è sufficiente in quanto esistono numeri composti (pochi) che verificano entrambi i predicati.

6.3.3 Lemma di Miller-Rabin (pochi composti verificano le CN)

Se N è numero composto il numero di testimoni y t.c. $2 \leq y \leq N - 1$ che soddisfa i predicati $P1$ e $P2$ è basso, precisamente $< \frac{N}{4}$.

Cioè Questo significa che se scelgo a caso un testimone y la probabilità che ne scelga uno che rende veri i due predicati è minore della probabilità di scegliere un testimone dove uno dei due predicati è falso.

$$P(\text{scegliere } y \text{ t.c. } P1 \text{ AND } P2 = 1) < \frac{\frac{N}{4}}{N-2} < \frac{1}{4}$$

Facciamo il rapporto tra casi favorevoli e casi possibili: $N - 2$ è il numero di casi possibili (scelte possibili tra i testimoni), $\frac{N}{4}$ sono i casi favorevoli (in cui i due predicati sono entrambi soddisfatti).

Quindi Immaginiamo i seguenti passi per verificare P1 e P2:

```
dato N scelgo a caso y in [2, N-1]
allora:
se uno dei due predicati è falso:
    N è certamente composto
se entrambi i predicati sono veri:
    N è composto con probabilità < 1/4, N è primo con probabilità > 3/4
```

Per ridurre la probabilità posso re-iterare il processo di selezione di y . Se ripeto il procedimento k volte scendiamo ad una probabilità $< \frac{1}{4^k}$ che N sia composto. Per avere un'idea iteriamo il processo trenta volte:

$$k = 30 \implies \left(\frac{1}{4}\right)^{30} \simeq 10^{-18}$$

6.3.4 Algoritmo completo e costo computazionale

Scriviamo l'algoritmo completo in pseudo-codice:

```
// Controlla la validita' del certificato y (certificato di N composto)
VERIFICA(N, y){
    if (P1 == false OR P2 == false) return 1
    return 0
}
TESTMR(N, k){
    for(i = 0; i < k; i++){
        y = numero a caso in [2, N-1]
        if(verifica(N, y) == 1) return 0 // N sicuramente non è primo
    }
    return 1 // N è primo con probabilità < 1/4^k
}
```

- Il costo di TESTMR è $k \cdot \text{VERIFICA}$ in quanto si itera la verifica sul singolo testimone per k volte. L'esecuzione k volte riduce la probabilità di errore da $\frac{1}{4}$ a $(\frac{1}{4})^k$
- **Calcoli per il primo predicato.** La verifica di $P1$ è il calcolo del MCD quindi si ha costo polinomiale nella dimensione di N tramite l'algoritmo di Euclide.

$$\text{MCD}(N, y) = 1$$

- **Calcoli per il secondo predicato.** La verifica di $P2$ richiede il calcolo di potenze

$$(y^z \bmod N = 1) \text{ OR } \left(\exists i : 0 \leq i \leq \omega - 1 \text{ t.c. } y^{2^i \cdot z} \bmod N = -1 \right)$$

La cosa è abbastanza critica perché queste operazioni richiedono tempo esponenziale nella dimensione di N . Si pensi che il valore massimo assumibile dall'esponente della y si ha con $i = \omega - 1$, quindi risulta necessario fare $\frac{N-1}{2}$ moltiplicazioni (otteniamo ciò da $N - 1 = 2^\omega z$, la premessa poco avanti) per ottenere il risultato desiderato

$$2^{(\omega-1) \cdot z} = \frac{N-1}{2} \implies \text{Caso peggiore: } y^{2^{(\omega-1) \cdot z}} = y^{\frac{N-1}{2}}$$

Risolviamo il problema implementando l'**algoritmo delle quadrature successive** (posto nell'appendice sull'algebra modulare), con cui riduciamo il numero di moltiplicazioni richieste per ottenere una potenza.

6.4 Generazione di numeri primi

Domanda da esame.

Illustrare con quale metodo si generano numeri primi grandi in crittografia e spiegare perché tale metodo è considerato efficiente.

Non abbiamo algoritmi propri per la generazione di numeri primi però possiamo seguire il seguente approccio:

1. generare un numero casuale
2. si testa la sua primalità
3. se questo numero casuale non è *dichiarato* primo si aumenta di due e si ripete da (2)

Densità dei numeri primi sull'asse degli interi L'algoritmo introdotto conviene perché per un lemma di Gauss sappiamo che il numero di interi primi e minori di un numero dato N tende a:

$$\longrightarrow \frac{N}{\log_e N} \text{ per } N \rightarrow \infty$$

quindi preso un N abbastanza grande sappiamo con quasi certezza che esisterà un primo in un intorno circolare di ampiezza $\log_e N$.

Algoritmo Scriviamo quindi un algoritmo in pseudocodice:

```
PRIMO(n): // n : numero di bit desiderati per il numero primo
    S = sequenza casuale di n-2 bit
    N = 1 S 1 // concatenamento dei bit per avere num. lunghi e sicuramente dispari
    while(TESTMR(N, K) == 0) N += 2
    return N
```

L'1 come bit più significativo serve per avere un numero sempre elevato (potrei avere sequenze con bit più significativi tutti uguali a 0), l'1 meno significativo mi dà certezza che andremo a considerare un numero dispari. Abbiamo quindi TESTMR con complessità $O(n^3)$, viene ripetuto $O(n)$ volte quindi in totale l'algoritmo ha complessità: $O(n^4)$.

Parte III

Cifrari storici

Capitolo 7

Introduzione

I cifrari sono nati per comunicazioni "sicure" ristrette a poche persone: la cosa si contrappone alla crittografia moderna e di massa, dove la sicurezza non è più legata alla non conoscenza dell'algoritmo da parte del crittoanalista. (quindi in piena contrapposizione alla crittografia moderna di massa). Distinguiamo i seguenti tipi di cifrari:

- cifrari a sostituzione, che possono essere monoalfabetici o polialfabetici;
- cifrari a trasposizione.

Vedremo alla fine la crittoanalisi statistica, utilizzata per forzare questi cifrari.

7.1 Principi di Ruggero Bacone

Tutti i cifrari storici rispettano i principi di *Ruggero Bacone* che dicono:

- cifratura e decifratura (funzioni C e D) devono essere eseguibili facilmente;
- deve essere impossibile decifrare il messaggio senza conoscere l'algoritmo (non posso ricavare la funzione D se la funzione C non è nota);
- il crittogramma deve apparire innocente (non deve destare sospetto, deve sembrare un messaggio in chiaro).

Capitolo 8

Cifrari a sostituzione

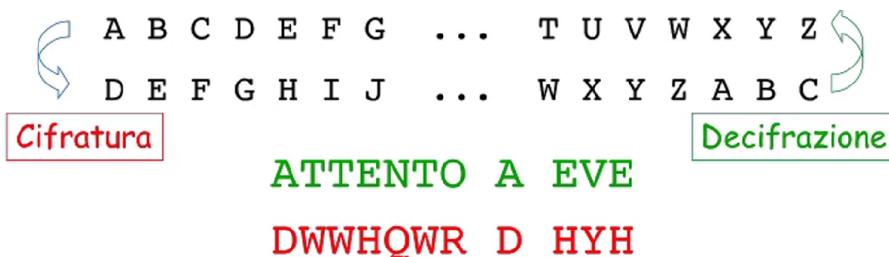
8.1 Tipologie

I cifrari a sostituzione si dividono in due classi:

- **monoalfabetici**, un carattere si sostituisce sempre con lo stesso carattere (per esempio il cifrario di Cesare);
 - Possibile impiegare funzioni di cifratura e decifrazione più complesse dell'addizione e della sottrazione in modulo.
 - Lo spazio delle chiavi è molto più ampio.
 - La sicurezza è comunque molto modesta.
- **polialfabetici**, un carattere si sostituisce con una lettera scelta da un insieme di lettere possibili, secondo una qualche regola (a seconda del contesto in cui appare la lettera).

8.2 Cifrario monoalfabetico: *cifrario di Cesare*

Il cifrario di Cesare è il primo di concezione moderna, tuttavia è senza chiave. La sua sicurezza si basa sulla ristrettezza di chi lo conosce. Si può generalizzare scegliendo un k arbitrario che indica di quanti posti dobbiamo ruotare l'alfabeto. Supponiamo $k = 3$, otteniamo



Si legge da sopra a sotto per cifrare, da sotto a sopra per decifrare.

Rafforzamento del cifrario Possiamo rafforzare il cifrario ponendo k come chiave: i due interlocutori decidono un valore $1 \leq k \leq 25$ (non 26, altrimenti il messaggio risultante è lo stesso posto in input).

Generalizzazione Indichiamo con $\text{pos}(x)$ la posizione di x nell'alfabeto

$$\text{pos}(A) = 0, \dots, \text{pos}(Z) = 25$$

per cifrare si deve quindi sostituire x con:

$$y : \text{pos}(y) = (\text{pos}(x) + k) \bmod 26$$

dove k è la chiave, $1 \leq k \leq 25$. Per decifrare si sostituisce y con:

$$x : \text{pos}(x) = (\text{pos}(y) - k) \bmod 26$$

Si può forzare facilmente (con attacco forza bruta) in quanto le chiavi sono poche.

Esempio Prendiamo $k = 10$. Vogliamo cifrare R

$$\text{pos}(R) = 17 \implies (17 + 10) \bmod 26 = 1 = \text{pos}(B)$$

Esercizio da esame, Decifrare il seguente crittogramma

YMNXJCJWHNXJNXJFXD

Deve essere trovata la chiave, facendo prove si trova che $k = 5$ (ho usato un sito online)

YMNXJCJWHNXJNXJFXD \rightarrow THISEXERCISEISEASY

Proprietà commutativa del cifrario Questo cifrario gode della proprietà commutativa, cioè data una sequenza di chiavi e di cifrature cambiando l'ordine in cui le eseguiamo non si varia il crittogramma finale. Questo significa che

$$\begin{aligned} C(C(s, k_2), k_1) &= C(s, k_1 + k_2) \\ D(D(s, k_2), k_1) &= D(s, k_1 + k_2) \end{aligned}$$

Comporre più cifrature non aumenta la sicurezza del sistema!

8.3 Cifrario monoalfabetico: *cifrario affine*

Una lettera in chiaro x viene sostituita con la lettera cifrata y che occupa nell'alfabeto la seguente posizione:

$$x : \text{pos}(y) = (a \cdot \text{pos}(x) + b) \bmod 26$$

Abbiamo come chiave segreta del cifrario $K = \langle a, b \rangle$. Mentre la cifratura rimane cosa semplice la decifrazione diventa più complessa.

$$y \rightarrow x : pos(x) = a^{-1} \cdot (pos(y) - b) \bmod 26$$

dove a^{-1} è l'inverso di $a \bmod 26$. Per maggiore chiarezza si invita a leggere le nozioni di algebra modulare.

Scelta della chiave Per far funzionare la decriptazione a e b vanno scelti in maniera particolare:

- b si può scegliere a piacere
- a va scelto invertibile cioè deve esistere...

$$a^{-1} : a \cdot a^{-1} \bmod 26 = 1$$

Generalizzando affermiamo che l'inverso di un intero a modulo m esiste ed è unico se e solo se i due numeri sono coprimi, cioè $\text{MCD}(a, 26) = 1$. Se il MCD è diverso da 1 allora non si ha l'iettività e non è possibile decifrare. Si veda il teorema dell'inverso nelle nozioni di algebra modulare per capire il perchè.

Esempio funzionante $K = \langle 3, 1 \rangle$

IL NOSTRO TEMPO \Rightarrow ZI OREGAR GNBUR

La chiave è valida poichè $\text{MCD}(3, 26) = 1$. Il fatto che 3 e 26 siano coprimi garantisce l'esistenza dell'inverso.

Esempio non funzionante

$$K = \langle 13, 0 \rangle$$

La chiave non è valida poichè $\text{MCD}(13, 26) \neq 1$.

- Tutte le lettere in posizione pari vengono trasformate in $A : pos(A) = 0$

$$A \Rightarrow pos(y) = (13 \cdot pos(A) + 0) \bmod 26 = 0 \bmod 26 = 0$$

$$C \Rightarrow pos(y) = (13 \cdot pos(C) + 0) \bmod 26 = 13 \cdot 2 \bmod 26 = 26 \bmod 26 = 0$$

$$E \Rightarrow pos(y) = (13 \cdot pos(E) + 0) \bmod 26 = 13 \cdot 4 \bmod 26 = 52 \bmod 26 = 0$$

$$G \Rightarrow pos(y) = (13 \cdot pos(G) + 0) \bmod 26 = 13 \cdot 6 \bmod 26 = 78 \bmod 26 = 0$$

- Tutte le lettere in posizioni dispari vengono trasformate in $B : pos(B) = 13$

$$B \Rightarrow pos(y) = (13 \cdot pos(B) + 0) \bmod 26 = 13 \bmod 26 = 13$$

$$D \Rightarrow pos(y) = (13 \cdot pos(D) + 0) \bmod 26 = 13 \cdot 3 \bmod 26 = 39 \bmod 26 = 13$$

$$F \Rightarrow pos(y) = (13 \cdot pos(F) + 0) \bmod 26 = 13 \cdot 5 \bmod 26 = 65 \bmod 26 = 13$$

$$H \Rightarrow pos(y) = (13 \cdot pos(H) + 0) \bmod 26 = 13 \cdot 7 \bmod 26 = 91 \bmod 26 = 13$$

Numero di chiavi Proviamo a contare le possibili chiavi tale che a e 26 siano coprimi. I fattori primi di 26 sono 2 e 13:

$$26 = 2 \cdot 13$$

Escludiamo i numeri pari (26 multiplo di 2) e 13 (non i multipli visto che non ci sono, agiamo modulo 26): il numero di valori assumibili da a è il seguente.

$$a \in \{\text{Numeri dispari tra 1 e 25 tranne 13}\} \rightarrow 12 \text{ valori possibili}$$

Mentre b può essere scelto con maggiore libertà, tenendo conto di un solo criterio:

$$b \text{ scelto a piacere tra 0 e 25} \rightarrow 26 \text{ valori possibili}$$

Se le chiavi legittime sono le coppie $\langle a, b \rangle$ allora abbiamo $12 \times 26 = 312$ chiavi (in realtà 311 visto che la coppia $\langle 1, 0 \rangle$ lascia inalterato il messaggio).

Segretezza della chiave Le chiavi sono troppo poche. Se la segretezza dipende unicamente dalla chiave allora **il numero delle chiavi deve essere così grande da essere immune da tentativi di ricerca esaustiva** e poi va scelto in maniera casuale. Come possiamo fare ciò in un cifrario monoalfabetico? Col prossimo cifrario.

8.4 Cifrario monoalfabetico: *cifrario completo*

Generiamo una permutazione a caso dell'alfabeto e la usiamo come chiave.

Lettera in chiaro di posizione $i \implies$ Lettera di posizione i nella permutazione

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

Quante possibili chiavi ci sono? Una chiave è una permutazione di 26 lettere, quindi lo spazio è $26! - 1$ ($4 \cdot 10^{26}$). Nel calcolo leviamo 1 per non considerare una corrispondenza dove nessun carattere risulta alterato.

Problema risolto? Questo grande spazio di chiavi tuttavia non è sufficiente a dire che il sistema è sicuro in quanto rimangono vulnerabilità basate su:

- strutture logiche dei messaggi in chiaro;
- occorrenza statistica delle lettere (osserviamo la lettera più frequente nel messaggio cifrato, questa potrebbe corrispondere alla lettera usata più frequentemente nell'alfabeto italiano).

Queste cose sono risolvibili solo passando ai cifrari polialfabetici.

8.5 Cifrario polialfabetico: archivio fotografico di Augusto

Il cifrario non veniva usato per comunicare ma per mantenere un archivio di informazioni cifrato. Esso fu svelato a seguito della sua morte dall'imperatore Claudio:

- Augusto scriveva i messaggi in greco, poi metteva in corrispondenza la sequenza di lettere del documento con la sequenza di lettere del primo libro dell'Iliade;
- sostituiva ogni lettera del documento con il numero che indicava la distanza tra le due lettere (quella nel messaggio in chiaro e quella nel primo libro dell'Iliade) nell'alfabeto greco.

Esempio

- **Lettera in posizione i nel documento:** α
- **Lettera in posizione i nell'Iliade:** ϵ
- **Carattere in posizione i nel crittogramma:** 4 (distanza tra α ed ϵ).

Forzatura Il cifrario è stato forzato perché Claudio ha trovato il libro dell'Iliade di Augusto con scritture, calcoli ed annotazioni. Il cifrario aumenta in sicurezza cambiando la pagina del libro ogni volta che si critta un nuovo documento.

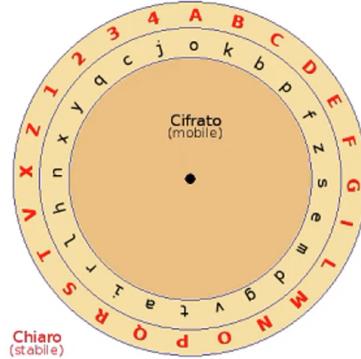
Perchè polialfabetico? Il cifrario è polialfabetico poichè tutte le α , ad esempio, non vengono sostituite sempre dal solito carattere. Dipende dalla posizione i -esima nel documento (nel caso di Augusto il primo libro dell'Iliade).

8.6 Cifrario polialfabetico: disco dell'Alberti (XV secolo)

Abbiamo due dischi ruotanti:

- quello esterno ha lettere (italiane) e numeri, e si usa per formulare il messaggio
- quello interno, più ricco (alfabeto inglese), ha lettere disposte in maniera arbitraria e diversa per ogni coppia di utenti

Ruotando uno dei due dischi è possibile ottenere corrispondenze diverse.



8.6.1 Primo metodo di cifratura

I dischi sono disposti in modo da ottenere la corrispondenza seguente...

ABCDEFGHIJKLMNPQRSTUVWXYZ12345 <-- CHIAVE A-S
SDTKBJOHRZCUNYEPXVFWAGQILM

ABCDEFGHIJKLMNPQRSTUVWXYZ12345 <-- CHIAVE A-Q
QILMSDTKBJOHRZCUNYEPXVFWAG

Chiave iniziale: A-S
messaggio: NON FIDARTI DI EVE
 $m = \text{NONFIDA } 2 \text{ RTIDIEVE}$
 $c = \text{UNUJRKS Q UYBMBSPS}$
quando si arriva su 2 la chiave diventa A-Q

Inizialmente abbiamo la chiave A-S, cioè indichiamo che tra i due dischi si debba avere corrispondenza tra A esterno ed S interno. Il numero due costituisce un carattere speciale, che determina un cambio di chiave: quando si arriva alla lettera Q si nota la corrispondenza con 2 e la chiave diventa A-Q (primo carattere con cui stabilisco la corrispondenza è sempre A, il secondo è quello che si trova nella corrispondenza con 2).

8.6.2 Secondo metodo di cifratura

C'è un secondo modo per usare questi dischi: il metodo dell' *indice mobile*:

ABCDEFGHIJKLMNPQRSTUVWXYZ12345 <-- CHIAVE A-E
EQHCWLMVPDNXAOGYIBZRJTSKUF

ABCDEFGHIJKLMNPQRSTUVWXYZ12345 <-- CHIAVE A-P
PDNXAOGYIBZRJTSKUFEQHCWLMV

$m: \text{ILD } 2 \text{ EL P FINO}$
 $c: \text{PDC S WD O OIRJ}$

Inizialmente la chiave è A-E, poi decifrando si trova 2: tra due lettere si cambia chiave. Si decifrano quindi altre 2 lettere nella giusta maniera e la lettera dopo sarà P cioè la nuova chiave A-P e successivamente si inizia a decifrare con questa nuova chiave. In genere si cambia chiave ogni volta che si trova un carattere speciale. Con caratteri speciali frequenti il cifrario è difficile da attaccare, poiché la chiave viene alterata ad intervalli imprevedibili.

8.7 Cifrario polialfabetico: De Vigenère

Esercizio da esame.

Esportare come funziona il cifrario di de Vigenère e descrivere il principale attacco che può essere condotto contro di esso.

E' una estensione della tecnica dell'Alberti più sicura che è rimasta irrompibile per 3 secoli. Si sceglie una chiave in cui ogni lettera corrisponde ad un numero che sarà di quanto bisogna shiftare il testo in chiaro per ottenere il testo cifrato: la chiave si ripete tante

C	H	I	A	V	E
2	7	8	0	24	4

volte quanto serve per equiparare il messaggio in lunghezza (nel caso in cui la lunghezza della chiave sia più piccola del messaggio da cifrare):

N	O	N	F	I	D	A	R	T	I	D	I	E	V	E
2	7	8	0	24	4	2	7	8	0	24	4	2	7	8
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
P	V	V	F	G	H	C	Y	B	I	B	M	G	C	M

Rappresentazione alternativa con matrice Si può vedere anche con una matrice 26x26 in cui ad ogni incrocio si trova la lettera i criptata con la lettera j . In sostanza indichiamo nella matrice non i numeri rappresentanti le traslazioni, ma direttamente i caratteri.

Sicurezza della chiave La sicurezza di questo metodo è influenzata dalla lunghezza della chiave: se la chiave è di piccola dimensione allora il numero di volte in cui si manifestano certa traslazioni aumenta. Addirittura, ponendo la chiave con dimensione $h = 1$ ottengo l'equivalente del cifrario monoalfabetico di Cesare. Maggiore è la lunghezza della chiave, maggiore è la sicurezza.

- **Idea di attacco.**

Supponiamo di conoscere la lunghezza h della chiave; costruiamo dei sottomessaggi formati dalle lettere che occupano tutte le stesse posizioni mod h . In ciascuno di questi messaggi tutte le lettere sono allineate alla stessa lettera della chiave quindi è come se fossero cifrate con un metodo monoalfabetico.

- **Sicurezza a confronto coi cifrari monoalfabetici.**

I cifrari polialfabetici non sono molto più sicuri dei metodi monoalfabetici se le chiavi sono piccole! Se la chiave è lunga quanto il messaggio, è casuale e non riutilizzata otteniamo un cifrario *perfetto* (ovviamente aumenta la difficoltà d'uso). E' il caso del *one-time-pad* che usa una codifica binaria (1917).

Capitolo 9

Cifrari a trasposizione

In questi cifrari non sostituiamo lettere come fatto fino ad ora. Il crittogramma presenta le seguenti caratteristica:

- è una permutazione dei caratteri del messaggio originario;
- in alcuni casi può includere lettere aggiuntive, ignorate nella decifrazione.

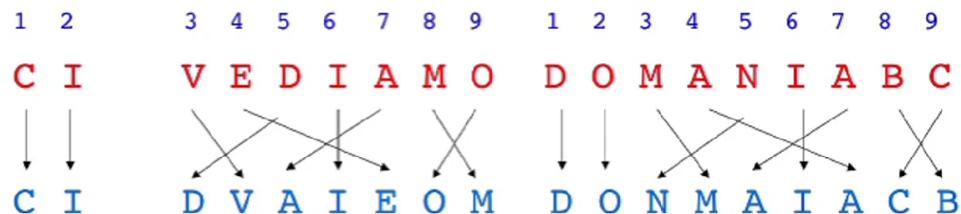
9.1 Cifrario a permutazione semplice

La chiave consiste nella coppia $\langle h, \Pi \rangle$.

- Dividiamo il messaggio da cifrare in blocchi aventi lunghezza h . Nel caso in cui il numero di caratteri che costituiscono il messaggio non sia multiplo di h andremo ad aggiungere nell'ultimo blocco dei *caratteri padding*, in modo che tutti i blocchi siano di dimensione h .
- Definiamo un insieme di posizioni Π , con cui rappresento una permutazione degli interi $\{1, \dots, h\}$ (ovviamente ho un numero di interi pari a h).

Vediamo col seguente esempio.

Esempio $h = 9, \Pi = \{1, 2, 5, 3, 7, 6, 4, 9, 8\}$



Numero di chiavi possibili Le chiavi sono le possibili permutazioni: $h! - 1$. Non si considera quella che lascia invariato il messaggio.

9.2 Cifrario a permutazione di colonne

La chiave consiste nella terna $k = \langle c, r, \Pi \rangle$

- c ed r sono il numero di colonne e righe di un tavolo da lavoro T dove andremo a collocare il messaggio in chiaro (di fatto il tavolo da lavoro è una matrice di r righe e c colonne).
- Π è la permutazione degli elementi $\{1, 2, \dots, c\}$. Ciò che andremo a spostare non sono caratteri, ma colonne del tavolo T !

Si prende il messaggio e lo si decompone in blocchi m_1, m_2, \dots di $c \times r$ caratteri ciascuno, eventualmente aggiungendo *padding*. I caratteri si distribuiscono tra le celle della matrice T , riempiendo le righe dall'alto verso il basso. Si veda il seguente esempio.

Esempio Vogliamo cifrare la frase *NON SONO IL COLPEVOLE*.

$$c = 6, r = 3, \Pi = \{2, 1, 5, 3, 4, 6\}$$

1	2	3	4	5	6
N	O	N	S	O	N
O	I	L	C	O	L
P	E	V	O	L	E

T

2	1	5	3	4	6
O	N	O	N	S	N
I	O	O	L	C	L
E	P	L	V	O	E

T permutata

Dalla lettura per colonne otteniamo il crittogramma *OIENOPPOOLNLVSCONLE*. In questo caso facciamo tutto in un unico blocco. Le chiavi sono esponenziali non avendo un tetto massimo per r e c .

9.3 Cifrario a griglia (*cifrario di Richelieu*)

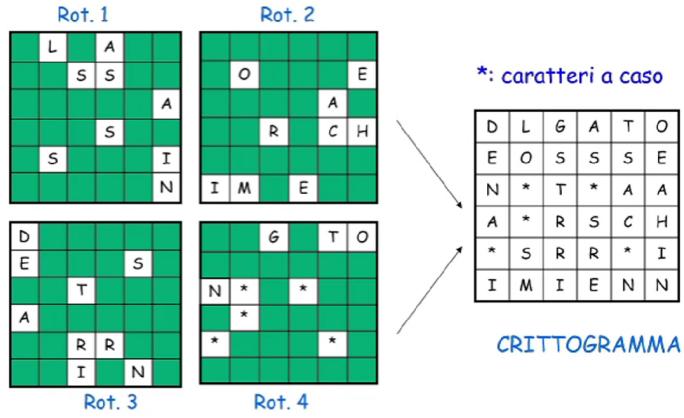
Domanda da esame.

Spiegare cosa s'intende per cifrario (storico) a griglia, indicare come si costruisce una griglia e quante griglie diverse si possono costruire per ogni dimensione scelta.

Versione innocente Un esempio è il cifrario di *Richelieu*: il crittogramma è celato in un libro, la chiave è una scheda perforata e la pagina di un libro. Facendo corrispondere la scheda con la pagina si vedono le lettere che compongono il messaggio. Il fatto che si prenda la pagina di un libro garantisce il rispetto del principio di innocenza di Bacone.

Versione non innocente Rinunciando al principio di innocenza usiamo una griglia quadrata $q \times q$ con q pari, $s = \frac{q^2}{4}$ è il numero delle celle trasparenti che compongono il messaggio. Si scrivono i primi s caratteri del messaggio nelle posizioni corrispondenti alle celle trasparenti. La griglia viene ruotata tre volte di 90° in senso orario e si ripete per ogni rotazione l'operazione di scrittura dei 3 gruppi successivi si s caratteri.

Esempio Vogliamo cifrare la frase *L'ASSASSINO E' ARCHIMEDES TARRINGTON*.



Abbiamo $q = 6, s = \frac{6^2}{4} = 9$. La griglia va costruita in modo che una cella già usata non ricapiti nuovamente. Se la lunghezza è maggiore di $4s$ si riempiono più tabelle. La decifrazione si ottiene applicando la maschera e leggendo.

Griglie possibili Le griglie possibili sono $G = 4^s = 4^{\frac{q^2}{4}}$. Per $q = 6$ otteniamo

$$G = 4^9 \approx 260'000$$

Si arriva a questo numero perché:

- immaginiamo di dover creare la maschera, scegliamo il primo foro e dobbiamo sceglierlo in modo che ruotando la griglia non sia nuovamente scoperto.
- si devono scegliere s buchi, per ogni cella se ne deve scegliere una su 4 quindi 4^s chiavi possibili

Capitolo 10

Forzatura dei cifrari storici

10.1 Crittoanalisi statistica

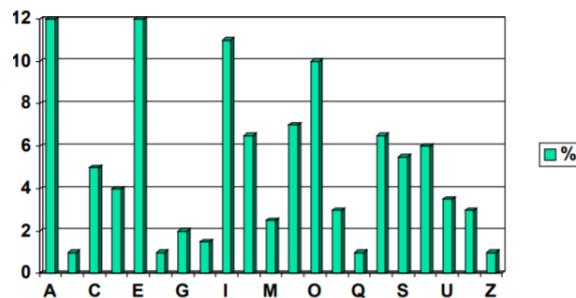
Domanda da esame.

Spiegare cosa s'intende per crittoanalisi statistica e come essa possa essere impiegata nell'attacco ai cifrari a sostituzione monoalfabetica e polialfabetica.

Abbiamo detto che la sicurezza di un cifrario dipende dalla dimensione delle chiavi, in modo tale da impedire la forzatura del cifrario stesso attraverso ricerche esaustive. Fino ad ora abbiamo pensato a forzature del cifrario con individuazione di chiave, ma il metodo non è l'unico possibile. I cifrari storici, in particolare, sono stati violati con un attacco statistico di tipo *cipher-text* (si conosce solo il crittogramma).

Ipotesi Il metodo si afferma in Europa nel XIX secolo con la forzatura del cifrario di Vigenère. Si fanno delle ipotesi:

- si suppone di conoscere il metodo di cifratura/decifratura (si da per scontato che il crittoanalista conosca il metodo);
- si suppone che il crittoanalista conosca il messaggio naturale in cui sono scritti i messaggi;
- si suppone di avere messaggi abbastanza lunghi da far valere le statistiche note dei linguaggi naturali.



La frequenza con cui appaiono le lettere dell'alfabeto è ben studiata in ogni lingua così come sono note le frequenze dei *digrammi*, *trigrammi*, *q-grammi* (gruppi da 2, 3, q lettere). Nell'italiano le lettere A ed E costituiscono il 12% dei messaggi. Si calcola il grafico delle frequenze dei caratteri all'interno del crittogramma: se siamo davanti ad un cifrario a sostituzione monoalfabetico avremo un grafico che è permutazione del grafico delle frequenze italiane, possiamo quindi dire che è molto probabile che se

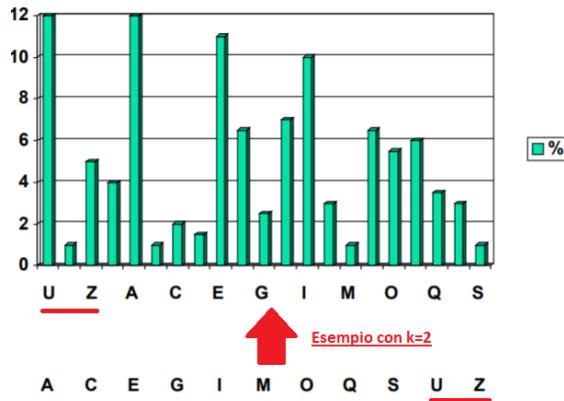
$$\text{freq}(x) \approx \text{freq}(y)$$

allora x è stato criptato con y .

10.2 Cifrari a sostituzione monoalfabetici

- **Cifrario di Cesare.**

Nel caso del cifrario di Cesare abbiamo uno shift del grafico. Ci basta individuare una corrispondenza per trovare tutte le altre.



L'attacco a forza bruta rimane la cosa più semplice, visto l'esiguo numero di chiavi.

- **Cifrario affine.**

Nel caso di cifrari affini una volta trovate due lettere a, b si imposta il sistema di equazioni e si risolve. Il cifrario è per definizione invertibile (basata avere $\text{MCD}(a, m) = 1$, quindi a, m coprimi), dunque la risoluzione del sistema è sempre possibile.

- **Cifrario completo.**

Nel caso di un cifrario completo (permutazione arbitraria) non abbiamo lo shift del grafico, ma una permutazione dove le frequenze non sono associate ai relativi caratteri. Quello che facciamo è associare le lettere del messaggio cifrato (l'unico che abbiamo) in base alle frequenze.

Esempio: so che una lettera (o un insieme di lettere) è quella maggiormente utilizzata nella lingua italiana. Verifico nel cifrario quale carattere è stato utilizzato maggiormente: confrontando le frequenze potrei individuare una corrispondenza.

10.3 Cifrari a sostituzione polialfabetica

Nella sostituzione polialfabetica la decifrazione è più difficile! A una lettera possono corrispondere lettere diverse: l'istogramma delle frequenze è piatto.

- **Cifrario dell'Alberti.**

Il cifrario dell'Alberti è immune da questi attacchi se la chiave viene cambiata spesso evitando *pattern* ripetitivi. Mantenere la stessa chiave a lungo è come usare un cifrario monoalfabetico, precisamente un cifrario completo.

- **Cifrario di Vigenère.**

- La lettera y non dipende solo dalla lettera del testo in chiaro, ma anche dalla lettera corrispondente della chiave.
- La chiave è spesso piccola e ripetuta più volte: se sappiamo che la chiave ha lunghezza h possiamo raggruppare le lettere in posizioni $h, 2h, 3h, \dots$ e poi $h+1, 2h+1, \dots$ e così via. In questo modo si ottengono h gruppi cifrati in maniera monoalfabetica: se individuo una coppia allora individuo tutte le altre coppie del gruppo analizzato.
- Si deve stimare la lunghezza della chiave: si studiano i digrammi ed i trigrammi alla ricerca di sequenze che si ripetono nella speranza che lo stesso digramma/trigramma sia criptato allo stesso modo e si prenda la loro distanza come multiplo della lunghezza della chiave (o la chiave stessa). E' molto più probabile che si generi in questo modo piuttosto che sia generato casualmente.

10.4 Cifrari a trasposizione

Nei cifrari a trasposizione non ha senso calcolare l'istogramma delle frequenze in quanto il crittogramma è una permutazione delle lettere del testo in chiaro (le frequenze sono le stesse). In questi casi si usa lo studio dei q-grammi. In particolare se si sa la lunghezza h della chiave:

- si divide il crittogramma in blocchi di h lettere
- in ciascun gruppo si cercano i gruppi di q lettere che formano i q-grammi più diffusi nel linguaggio
- se si trova un vero q-gramma si ottiene parte della chiave

Osservazione conclusiva.

La crittoanalisi statistica permette di fare due cose:

- individuare il possibile cifrario adottato da chi ha generato il crittogramma;
- individuare (in alcuni casi) le corrispondenze, cioè decifrare in tutto o in parte il crittogramma.

Capitolo 11

La macchina Enigma

Rappresenta l'ultimo esempio di cifrario storico, il primo che porta a dei sistemi automatizzati e gli sforzi fatti per rompere questo sistema sono stati importanti per l'evoluzione storica dell'informatica. Nasce in Germania nel 1918 ed è una modifica automatizzata del cifrario dell'Alberti.



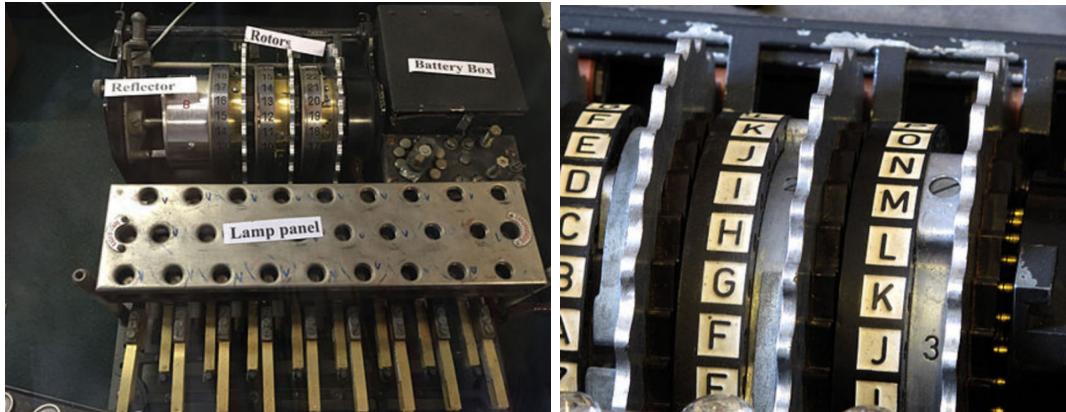
Si compone di una tastiera di ventisei lettere (alfabeto tedesco) più altrettante lampadine rappresentanti lettere. Ogni volta che l'utente preme un tasto si illumina il corrispondente carattere del crittogramma. L'utente osserva la luce accesa, si segna la lettera e prosegue col carattere successivo. Lo stesso approccio vale per la decifrazione. Si osservi che nella cifratura premere più volte lo stesso carattere sulla tastiera porta ad ottenere un crittogramma costituito da caratteri diversi.

11.1 Chiave e assetto iniziale

La chiave è l'assetto iniziale della macchina. Questo assetto rende reversibile la cifratura: un assetto permette di cifrare ($A \rightarrow F$) ma anche di decifrare ($F \rightarrow A$). Ciascun interlocutore presenta una macchina enigma: affinchè il destinatario possa decifrare un messaggio è necessario che tutti gli interlocutori abbiano lo stesso assetto iniziale.

11.2 Interno della macchina

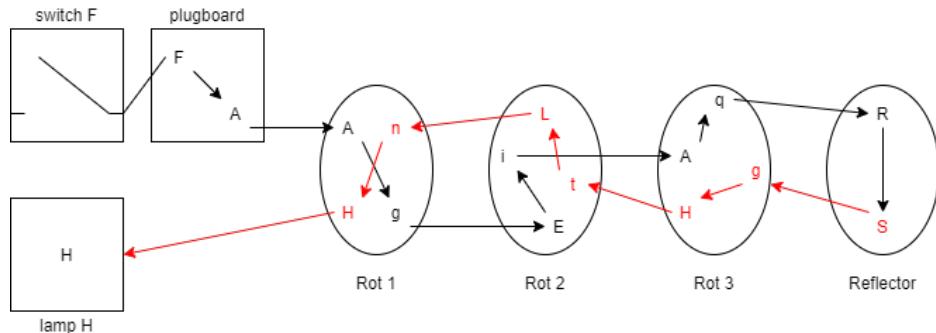
All'interno abbiamo un *riflettore* e 3 rotori (dischi in grado di ruotare in modo indipendente).



Il riflettore è anch'esso un rotore modificato. Per aumentare lo spazio delle chiavi si aggiunge una *plugboard* cioè una batteria di connettori che mette in corrispondenza diretta due lettere (in un certo senso è come se scambiassi di posizione due lettere, premo una e in realtà ne sto premendo un'altra).



Ogni rotore rappresenta la permutazione di un alfabeto. Possiede due facce : *pad* e *pin*, ciascuna con ventisei contatti elettrici disposti in cerchio e ciascuno associato a una particolare lettera. Si ha un cablaggio fisso che collega i contatti di una faccia con quelli di un'altra.



- Per costruzione una lettera non è mai cifrata con se stessa. Questa proprietà e quella di riflessione sono state utilissime nel rompere questo sistema.
- I rotori non sono fissi.
 - Il primo rotore avanza di un passo per ogni lettera battuta sulla tastiera.
 - Dopo 26 passi il primo rotore torna sulla posizione iniziale, quindi si muove di una posizione il secondo rotore.
 - Dopo la rotazione completa del secondo rotore si muove di un passo il terzo rotore.

Questa cosa comporta una chiave cambiata ad ogni passo.

- Le permutazioni sono 26 per il primo rotore rispetto al secondo, 26 del secondo rispetto al terzo, 26 del terzo rispetto al riflettore:

$$26 \cdot 26 \cdot 26 = 17.576$$

chiavi se i rotori sono immutabili e noti a chiunque ne avesse una copia.

- Si aumentano le chiavi scambiando l'ordine dei rotori, quindi si sale a $26^3 \cdot (3!) > 10^5$.
- Aggiungendo la plugboard si possono scambiare 6 coppie di caratteri per ogni trasmissione, si arriva quindi ad una sequenza di 12 caratteri per descrivere il cablaggio: le combinazioni possibili sono

$$\binom{26}{12} \approx 10^7$$

Scelte le lettere va scelta la loro ordinazione:

$$\binom{26}{12} \cdot 12!$$

In realtà dobbiamo escludere delle permutazioni, quelle che lasciano lo stesso effetto, quindi dividiamo per $6!$ e per 2^6

- Ogni cablaggio è descritto da una sequenza di 12 caratteri (le 6 coppie da scambiare)
- Combinazioni possibili: $\binom{26}{12} \sim 10^7$
- Ogni gruppo di 12 caratteri si può presentare in $12!$ permutazioni diverse, ma non tutte producono effetti diversi:

AB CD EF GH IJ KL

CD AB EF GH IJ KL

producono lo stesso effetto, e con queste anche tutte le 6! permutazione delle 6 coppie

Infine si devono considerare i possibili scambi tra gli elementi delle coppie, che producono lo stesso effetto

AB CD EF GH IJ KL

BA CD EF HG IJ KL

Dobbiamo dividere per un ulteriore fattore $2^6 = 64$

Parte IV

Cifrari perfetti

Capitolo 12

Introduzione

I cifrari perfetti sono cifrari che offrono una **sicurezza incondizionata**, proteggono le informazioni con certezza assoluta di fronte a qualsiasi potenza di calcolo. Solo chi è in possesso della chiave può decifrare. Un attacco a forza bruta non può rompere la cifratura (pur avendo risorse computazionali spropositate).

Costo e crittografia di massa Si paga un caro prezzo nella loro implementazione, infatti questi cifrari sono usati solo in pochi ambiti, per la crittografia di massa si preferisce avere una sicurezza computazionale scommettendo su $P \neq NP$.

Definizione informale Questo concetto è stato formalizzato da Shannon (nel 1949, in realtà prima) informalmente un cifrario è perfetto se la sicurezza è garantita qualunque sia l'informazione carpita dal canale.

12.1 Formalizzazione del concetto di *cifrario perfetto*

Abbiamo lo spazio dei messaggi MSG e lo spazio dei crittogrammi $CRITTO$. Abbiamo poi le variabili aleatorie $M \in MSG$, che descrive il comportamento del mittente, e $C \in CRITTO$, che descrive il processo di comunicazione sul canale.

- Ricorriamo alla teoria della probabilità per definire:
 - $P(M = m)$: probabilità che il mittente voglia spedire il messaggio m al destinatario
 - $P(M = m|C = c)$: probabilità condizionata (a posteriori) che il messaggio inviato sia effettivamente m , dato che sul canale transita il crittogramma c
- Si suppone che il crittoanalista conosca tutto il sistema tranne la chiave. Conosce:
 - distribuzione di probabilità con cui il mittente invia un messaggio
 - conosce il cifrario
 - conosce lo spazio delle chiavi K

Definizione Un cifrario è perfetto se $\forall m \in MSG, \forall c \in CRITTO$ vale:

$$P(M = m | C = c) = P(M = m)$$

Cioè: la conoscenza di C non ci permette di dire nulla sul messaggio.

12.1.1 Esempi di cifrari non perfetti (casi estremi)

Supponiamo che ci sia un messaggio \bar{m} che il mittente invia con probabilità $0 < p < 1$.

$$P(M = \bar{m}) = p$$

Cosa succede in assenza dell'uguaglianza alla base della definizione di cifrario?

Esempio estremo 1 Supponiamo di avere probabilità uguale ad 1.

$$\exists \bar{m}, \bar{c} : P(M = \bar{m} | C = \bar{c}) = 1$$

Il caso è estremo. Con probabilità 1 sicuramente il messaggio associato al crittogramma \bar{c} è \bar{m} , dunque aumenta la nostra conoscenza sul sistema.

Esempio estremo 2 Supponiamo una situazione dove la probabilità è nulla

$$\exists \bar{c} : P(M = \bar{m} | C = \bar{c}) = 0$$

ci permette di dire che se passa \bar{c} non è stato sicuramente spedito il messaggio m . Anche qui la nostra conoscenza aumenta: il sistema non è perfetto!

Quindi In un cifrario perfetto la conoscenza complessiva del crittoanalista non cambia dopo che è stato osservato un crittogramma in transito:

m e c sono del tutto scorrelati, c appare essere una sequenza casuale

12.2 Teorema di Shannon

In un cifrario perfetto il numero delle chiavi deve essere maggiore o uguale al numero dei messaggi possibili

Dimostrazione Vogliamo dimostrare che $N_k \geq N_m$

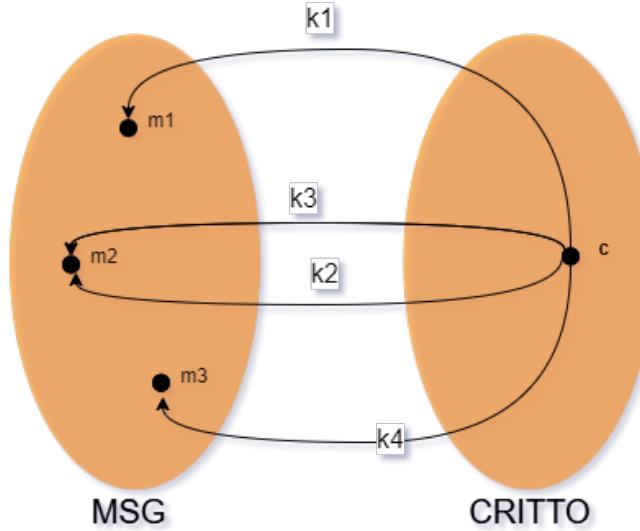
$$\begin{aligned} N_m &= \#\{m \in MSG : P(M = m) > 0\} \quad \text{Num. dei messaggi possibili} \\ N_k &= \#\{\text{insieme delle chiavi}\} \end{aligned}$$

Supponiamo per assurdo che $N_k < N_m$, e prendiamo un crittogramma c tale

$$P(C = c) > 0$$

Cerchiamo di contare quanti messaggi corrispondono a questo crittogramma.

- Supponiamo che corrispondano S messaggi. Essi sono i messaggi che posso ottenere decifrando c con tutte le chiavi possibili. Decifro con la chiave k_1 e ottengo m_1 , decifro con la chiave k_2 e trovo m_2 , decifro con la chiave k_3 e ottengo sempre m_2 (non lo posso escludere), e così via...



- Si ha necessariamente $S \leq N_k$ (posso ottenere al più N_k messaggi, non di più).
- Poichè abbiamo iniziato con l'ipotesi $N_k < N_m$ otteniamo

$$S \leq N_k < N_m$$

quindi

$$S < N_m$$

Il numero di messaggi associati al crittogramma c (S) è strettamente minore del numero di messaggi possibili.

- Appurato che $S < N_m$ affermiamo che deve esistere un messaggio m con probabilità $P(M = m) > 0$ tale che

$$P(M = m \mid C = c) = 0$$

decifrando c esisterà un m che sicuramente non è il messaggio.

- Inferisco della conoscenza, quindi contraddico il cifrario perfetto.** Ne deriva quindi che $N_k \geq N_m$. Per usare un cifrario perfetto mi serve quindi una chiave lunghissima!

■

Capitolo 13

One-Time Pad

Domanda da esame. Qual è lo svantaggio principale del cifrario One-Time Pad?

13.1 Cifratura e decifratura

Il *One-Time Pad* è un cifrario precedente Shannon, simile ad un Vigenère con chiave lunga quanto il messaggio ma su un alfabeto binario. Nasce nel 1917 da Mauborge e Vernam. *Pad* significa *blocco*, *One-Time* poichè la chiave può essere utilizzata una sola volta.

Ragionamento Come alfabeto si usa $\{0, 1\}$. MSG, CRITTO, KEY sono spazi delle sequenze binarie e l'algoritmo di cifratura è noto a tutti ed è lo XOR (somma modulo 2). Supponiamo quindi $m, k \in \{0, 1\}^n, n > 0$:

$$\begin{aligned} c &= C(m, k) = m \oplus k \\ m &= D(c, k) = c \oplus k \end{aligned}$$

Esempio Dato $n = 5, m = 10110, k = 01011$ otteniamo

$$c = 11101$$

Se io riapplico nuovamente lo XOR ottengo di nuovo

$$m = 10110$$

Questo ci torna poichè

$$m = c \oplus k = m \oplus k \oplus k = m \oplus 0 = m$$

Necessità di cambiare la chiave La chiave deve essere cambiata frequentemente. Supponiamo di non farlo

$$c_1 = m_1 \oplus k$$

$$c_2 = m_2 \oplus k$$

Proviamo a fare il seguente calcolo

$$c_1 \oplus c_2 = m_1 \oplus k \oplus m_2 \oplus k = m_1 \oplus m_2$$

non ottengo la chiave, ma so che l'operatore XOR restituisce lo zero se due caratteri sono uguali. Il crittoanalista ha imparato qualcosa di nuovo.

Sicurezza L'unica informazione ottenibile dal one-time-pad è **la lunghezza del messaggio**. La sicurezza del One-Time Pad non è computazionale: l'attacco forza bruta non ha alcun senso perchè da tutte le chiavi possibili otteniamo un vasto insieme di messaggi possibili. Mittente e destinatario, a volte, si mandano messaggi privi di senso per rafforzare la sicurezza.

13.2 Teorema su *One-time pad* perfetto e minimale

One-Time Pad è un cifrario perfetto e minimale (usa un numero minimo di chiavi) se:

- tutti i messaggi hanno lunghezza n , se più corti li paldo, se più lunghi li divido in blocchi;
- tutte le sequenze di n bit hanno probabilità maggiore di zero di essere inviate (tutte le sequenze di bit sono messaggi, ogni tanto mittente e destinatario si scambiano sequenze prive di significato);
- si usano chiavi scelte perfettamente a caso per ogni messaggio.

Chiaramente la cosa è complessa: la chiave deve essere lunga quanto il messaggio e deve essere cambiata ogni volta.

Si ricordi la definizione di probabilità condizionata:

$$P(M = m | C = c) \triangleq \frac{P(M = m \text{ e } C = c)}{P(C = c)}$$

Al numeratore abbiamo la probabilità che entrambi gli eventi siano avvenuti: che Alice invii il messaggio m e che l'abbia cifrato ottenendo il crittogramma c . Si divide per la probabilità che stia transitando c sul canale di comunicazione.

Domanda da esame. Spiegare con precisione matematica e proprietà di linguaggio perché il cifrario One-Time Pad su messaggi di n bit non può essere ritenuto perfetto se la chiave non è scelta perfettamente a caso

Dimostrazione perfettezza

- **Tesi.** La tesi è la definizione di cifrario perfetto:

$$\forall m \in MSG, \forall c \in CRITTO$$

$$P(M = m | C = c) = P(M = m)$$

- Stimiamo il numeratore. Dato un messaggio m vogliamo ottenere un crittogramma c : per le proprietà dello XOR esiste una sola chiave k in grado di farmi ottenere c a partire da m

$$\exists! k \in KEY : m \oplus k = c$$

Segue che la probabilità di ottenere il crittogramma c dal messaggio m è pari alla probabilità di scegliere casualmente la chiave k : $\frac{1}{2^n}$ (formulina già trovata, quando abbiamo introdotto la casualità)

$$\forall c \in CRITTO \quad P(C = c) = \frac{1}{2^n}$$

Nella formula non abbiamo alcuna dipendenza della probabilità dal messaggio (si pensi all'ipotesi 3 del teorema), quindi possiamo scrivere la probabilità nel seguente modo

$$P(C = c \text{ e } M = m) = P(M = m) \cdot P(C = c)$$

- Prendiamo la definizione di probabilità condizionata e sostituiamo

$$P(M = m | C = c) = \frac{P(M = m \text{ e } C = c)}{P(C = c)} = \frac{P(M = m) \cdot \cancel{P(C = c)}}{\cancel{P(C = c)}}$$

siamo arrivati alla definizione di cifrario perfetto!

■

Dimostrazione minimale Sappiamo da Shannon che il numero delle chiavi deve essere maggiore o uguale rispetto al numero dei messaggi

$$N_k \geq N_m$$

ma nel one-time pad su lunghezza n ho 2^n messaggi possibili, e 2^n crittogrammi possibili

$$N_k = N_m = N_{CRITTO} = 2^n$$

Anche le chiavi sono sequenze di bit e ne uso il numero minore possibile. Ricordiamo che tutte le sequenze di n bit sono messaggi possibili: l'attacco forza bruta non ha senso.

■

13.3 Riduzione della dimensione della chiave

Cerchiamo di ridurre la dimensione delle chiavi. Pensiamo alla lingua inglese, dove i messaggi significativi sono solo α^n con $\alpha = 1.1$

$$\alpha^n << 2^n$$

Poniamo $N_m = \alpha^n$ dato che $N_k = 2^t \geq N_m = \alpha^n$. Vogliamo individuare il numero di bit t , necessari affinchè

$$\begin{aligned} 2^t \geq \alpha^n &\longrightarrow t \cdot \log_2 2 \geq \log_2 \alpha^n = n \cdot \log_2 \alpha \\ t &\geq \log_2 \alpha^n = n \cdot \log_2 \alpha \longrightarrow t \geq 0.12 \cdot n \end{aligned}$$

Posso quindi usare chiavi molto più corte: $\approx 10\%$ di n .

Come fare? Genero i t bit randomicamente e li estendo in maniera deterministica su n bit (funzione che dato in ingresso una sequenza su t bit mi restituisce una sequenza su n bit, sempre lo stesso output con lo stesso input). Dobbiamo però metterci al riparo dall'attacco forza bruta: è fondamentale che coppie diverse di (m, k) producano lo stesso crittogramma. Per far ciò il $\#(m, k)$ deve essere di molto maggiore di $\#CRITTO$:

$$\alpha^n \cdot 2^t >> 2^n \rightarrow t >> 0.88n$$

In definitiva ci serve $t >> 88\%n$ quindi non si ha una grande compressione della chiave.

Parte V

Cifrari simmetrici moderni

Capitolo 14

Data Encryption Standard (DES)

14.1 Introduzione e principi di Shannon

Il *Data Encryption Standard* è un cifrario simmetrico che è stato lo standard per la crittografia di massa fino all'avvento dell'AES. Pur non essendo più usato è importante conoscerlo: i cifrari moderni si basano sulla struttura del DES. Il DES è il primo algoritmo a basarsi sui principi di Shannon.

- **Diffusione:** ogni singolo carattere del crittogramma dipende da tutti i caratteri del testo in chiaro.
- **Confusione:** combinare testo in chiaro e chiave in modo complesso in modo che osservare il crittogramma non possa portare a separare le due sequenze (testo in chiaro e chiave).

Questi principi sono importanti per la resistenza agli attacchi di crittoanalisi statistica.

14.2 Storia

Nasce nel 1972 quando NBS (*National Bureau of Standard*) ora NIST (*National Institute for Security and Technology*) chiese la creazione di un algoritmo di crittografia simmetrica standard (standard poichè ogni compagnia usava un proprio sistema crittografico).

Richieste dell'ente In particolare le richieste erano:

- sicurezza basata sulla segretezza della chiave e non sul processo di cifratura e decifrazione (tutto pubblico tranne la chiave);
- l'algoritmo doveva essere efficiente sia in software che in hardware;
- la sua sicurezza doveva essere certificata da terzi (un terzo che garantiva la sicurezza del sistema, con sistemi proprietari rimane il dubbio sull'affidabilità).

Il primo bando andò deserto, al secondo l'IBM propose *Lucifer* e lo lasciò studiare alla NSA che introdusse alcune variazioni:

- riduzione della dimensione della chiave da 128 bit a 56 bit;
- modifiche nella la S-box (che vedremo a breve, componente cruciale del cifrario).

La IBM sospettò che la NSA volesse rendere più facile la rottura del cifrario, ma alla fine accettò le modifiche dopo averle studiate a fondo.

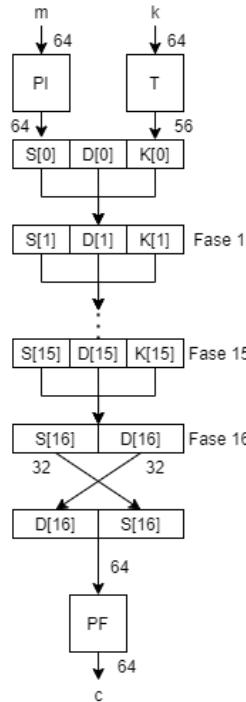
Pubblicazione e rinnovo periodico Il cifrario è stato reso pubblico nel 1977 con licenza d'uso gratuito. La certificazione era rinnovata periodicamente valutando le evoluzioni nella crittoanalisi. Quando si trova un attacco con costo inferiore a quello di un attacco forza bruta (sullo spazio delle chiavi) allora il cifrario si dice forzato.

3-DES (triplo DES) E' rimasto in vita fino al 1999 quando ne è stato sconsigliato l'uso per una versione più aggiornata: il 3-DES.

AES Nel 2005 anche il 3-DES diventa sconsigliato a fronte dell'AES proposto nel 2000 ed entrato nel 2001 all'utilizzo di massa. Ad oggi l'AES non è stato ancora rotto.

14.3 Cifratura

Nel DES la cifratura avviene per blocchi di 64 bit, la chiave è di 64 bit in cui 56 sono casuali ed 8 sono di parità (ogni 7 un bit di parità, serve per verificare che la chiave venga acquisita in modo corretto). La cifratura si compone di $r = 16$ fasi in cui si ripetono le stesse operazioni.



Nel grafico individuiamo:

- m: blocco del messaggio (abbiamo in ingresso il blocco a 64 bit da criptare)
- k: chiave segreta con i bit di parità (64 bit, di cui 8 bit di parità)
- PI e PF: permutazione iniziale e finale (non ci sono sempre)
- c: corrispondente blocco del crittogramma (risultato finale dopo la 16-esima fase)

Abbiamo in ingresso il messaggio m a 64 bit e la chiave a 64 bit (di cui otto di parità). Permutiamo i bit del messaggio con PI , permutiamo anche i bit della chiave con T ottenendo $K[0]$. Attenzione: T restituisce solo 56 bit perché scarta i bit di parità (di cui si conosce la posizione). A questo punto dividiamo il messaggio m permutato (eventualmente permutato) in due blocchi: $S[0]$, $D[0]$. Eseguiamo una serie di fasi e alla fine, dopo l'ultima, scambiamo di posizione i blocchi $S[16]$ e $D[16]$. Eventualmente permutiamo.

Funzioni In ogni fase $i = 1, 2, \dots, 16$ andiamo ad applicare le seguenti funzioni:

$$\begin{aligned} S[i] &= D[i-1] \\ D[i] &= S[i-1] \oplus f(D[i-1], K[i-1]) \end{aligned}$$

Si scambiano le due metà dell'input, e D viene ottenuto con l'operatore XOR bit a bit tra $S[i-1]$ e una funzione non lineare $f(D[i-1], K[i-1])$ (la S-box detta prima). Con le permutazioni e gli scambio realizzo la **diffusione** (lo vediamo perchè alterando un solo bit del messaggio in chiaro si ha un drastico cambiamento nell'intero crittogramma), con la S-box la **confusione** (la S-box prende in ingresso la chiave).

PI, PF e T PI, PF e T sono delle tabella che vanno lette per riga:

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

40	1	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31	
38	6	46	14	54	22	62	30	
37	5	45	13	53	21	61	29	
36	4	44	12	52	20	60	28	
35	3	43	11	51	19	59	27	
34	2	42	10	50	18	58	26	
33	1	41	9	49	17	57	25	

57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	52	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

Permutazione PI

Permutazione PF

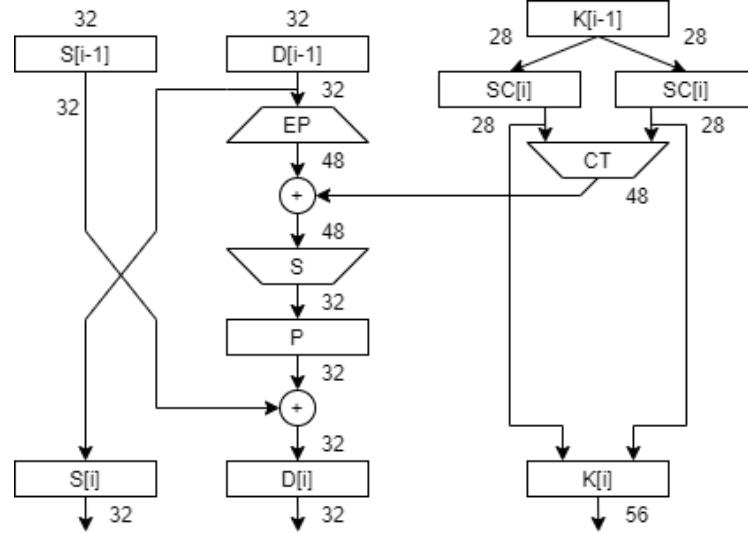
Trasposizione T

Le tabelle vanno lette per righe. La permutazione PI riordina i bit del messaggio $m = m_1 m_2 \dots m_{64}$ come $m_{58} m_{50} \dots m_7$ (porta in posizione 40 il bit in posizione 1)

PF è la permutazione inversa di PI, cioè riporta in posizione 1 il bit in posizione 40, etc.

T provvede anche a scartare dalla chiave $k = k_1 k_2 \dots k_{64}$ i bit per il controllo di parità $k_8, k_{16}, \dots, k_{64}$, generando una sequenza di 56 bit che costituisce la prima sottochiave $k[0]$.

Fase La fase i -esima del DES è rappresentata dal seguente schema.



Abbiamo come input i blocchi $S[i - 1]$, $D[i - 1]$ (ottenuti dalla fase precedente, o i blocchi iniziali posti come indicato prima) e $K[i - 1]$ (la chiave, alterata ad ogni step). La chiave viene divisa in due metà da 28 bit ciascuno. Su ciascuna metà si esegue uno shift ciclico (SC) a sinistra, dove il numero di posizioni dipende dall'indice della fase:

- 1 se $i \in \{1, 2, 9, 16\}$
- 2 altrimenti

A questo punto i blocchi ottenuti vengono usati in due modi:

- concatenati e posti come $K[i]$ (chiave usata nel prossimo step)
- concatenati e posti in ingresso in CT .

Il blocco CT permuta e scarta alcuni bit (si passa da 56 a 48 bit). Per quanto riguarda le due sequenze da 32 bit ($S[i - 1]$ e $D[i - 1]$):

- pongo in $S[i]$ il contenuto di $D[i - 1]$

$$S[i] = D[i - 1]$$

- pongo i bit di $D[i - 1]$ in ingresso ne blocco EP, che permuta i bit e li espande (duplicazione di alcuni bit, si passa da 32 bit a 48 bit);
- applico lo XOR alle due sequenze a 48 bit (quella ottenuta da EP e quella ottenuta da CT)
- applico la S-box (che garantisce la non linearità della funzione, si passa da 48 a 32 bit);
- permuto i 32 bit ottenuti dalla S-box e applico l'operatore XOR tra questi ed $S[i - 1]$.

Funzioni CT ed EP

14	17	11	24	01	05
03	28	15	06	21	10
23	19	12	04	26	08
16	07	27	20	13	02
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

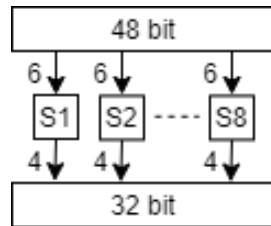
La funzione CT

otto bit dell'ingresso
(e.g., il bit 09) non sono
presenti in uscita.

La funzione EP

sedici bit di ingresso sono
duplicati (e.g., il bit 32 è
copiato nelle posizioni 1 e 47
dell'uscita).

S-Box La S-box implementa 8 funzioni booleane a 6 input e a 4 output (in bit).



Queste funzioni sono rappresentate attraverso delle tavola di verità, che vanno però lette in maniera particolare (di seguito la tabella di S1). Prendiamo il seguente esempio...

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

Supponiamo di avere 010011: prendo gli estremi 0 ed 1 ed uso questo come indice per la riga, prendo poi i 4 bit centrali (1001) e lo uso come indice di colonna. Accedo quindi a $(1, 9) \rightarrow 06$.

Perché è importante che non sia lineare? Perché

$$f(x \oplus y) \neq f(x) \oplus f(y)$$

ed è cruciale ai fini del funzionamento!

14.4 Proprietà del DES con m, c complementati

Consideriamo i crittogrammi c, c^* , con il messaggio m e la chiave k :

$$c = C_{DES}(m, k) \quad c^* = C_{DES}(\bar{m}, \bar{k})$$

Abbiamo visto che prima della S-box applichiamo l'operatore XOR (ricordarsi $x \oplus 1 = \bar{x}$)

$$\bar{m}_i \oplus \bar{K}_i = (1 \oplus m_i) \oplus (1 \oplus K_i) = m_i \oplus K_i$$

Date le coppie $\langle m, k \rangle, \langle \bar{m}, \bar{k} \rangle$ l'input della S-box **è lo stesso**, dunque dalla S-box esce lo stesso output. In realtà il crittogramma restituito è il complemento

$$c^* = \bar{c}$$

Questo perchè nello XOR finale si ha complementazione in uno solo dei due blocchi (abbiamo un blocco posto direttamente che può essere complementato o no e un altro da cui otteniamo la stessa sequenza - *S-box* - indipendentemente dall'essere complementato o no).

Esempio

8-bit binary Plaintext atau Ciphertext	8-bit binary Plaintext atau Ciphertext
11111111	00000000
1010101010	0101010101
Hasil	Hasil
0 0 0 0 1 0 0 0	1 1 1 1 0 1 1 1

<https://fauzanakmalh1.github.io/Simplified-DES-Calculator/>

14.5 Attacchi al DES

Gli attacchi al DES sono tutti attacchi a forza bruta:

- **architetture appositamente progettate per attaccare il DES** e quindi velocizzare criptazione e decriptazione (con 1M\$ si costruiva una macchina che forzava il DES in 35 minuti);
- **distribuire lo spazio delle chiavi tra più utenti**, è più economico e la velocità dipende da quanti utenti vengono coinvolti.

challenges di RSA Nel 1997 la compagnia RSA ha offerto una ricompensa di dieci-mila dollari a chi avrebbe decifrato un crittogramma (non tanto trovare il messaggio, ma la chiave che ha prodotto quel particolare crittogramma attraverso il DES): in 5 mesi esplorando il 25% delle chiavi viene risolta la sfida e trovata la chiave. Nel 1998 è stata lanciata una seconda sfida: ci vollero 39 di giorni esplorando 85% delle chiavi.

1997 : strong cryptography makes the world a safer place

1998 : many hands make light work

14.5.1 Attacchi esaurenti

Le chiavi possibili sono 2^{56} , 64 di queste chiavi sono rimosse perché con regolarità. Ricordando che:

$$C(m, k) = c$$

$$\bar{c} = C(\bar{m}, \bar{k})$$

si possono dimezzare le chiavi passando da 2^{56} a 2^{55} ($2^{56} = 2 \cdot 2^{55}$, 55 bit di sicurezza). Questo è un tipo di **attacco chosen plain-text** (scelto perché i messaggi in chiaro devono avere una certa relazione) in cui il crittoanalista si procura delle coppie

$$\langle m, c_1 \rangle, \langle \bar{m}, c_2 \rangle$$

Si inizia ad esplorare le chiavi e per ognuna si controlla:

- $C(m, k) = c_1$: k probabilmente è la chiave
- $C(m, k) = \bar{c}_2$: \bar{k} probabilmente è la chiave
- Se $c_1 \neq \bar{c}_2$ si prova un'altra chiave (k e \bar{k} non lo sono)

Possiamo dire

$$C(m, k) = \bar{c}_2 \iff C(\bar{m}, \bar{k}) = \bar{c}_2 = c_2$$

Escludo quindi due chiavi alla volta ($C(m, k)$ si calcola una volta soltanto, mentre la complementazione è operazione "tranquilla").

14.5.2 Crittoanalisi differenziale

Nel 1990 sono stati scoperti attacchi di *crittoanalisi differenziale* (da Biham e Shamir).

- E' un attacco di tipo **chosen plain-text** in cui il crittoanalista si procura 2^{47} coppie $\langle m, c \rangle$ (messaggio scelto dal crittoanalista, obv).
- I messaggi sono scelti in modo tale da avere differenze particolari, il crittoanalista va a verificare le variazioni nel crittogramma. Si sfruttano le similitudini per arrivare alla chiave.
- Si assegnano delle probabilità alle singole chiavi, ed emerge poi quella più probabile.

Costo Dato che le fasi sono 16 si ha un costo di questo attacco pari a $2^{55.1}$, poco più di una ricerca esaustiva (quindi nulla di interessante sul piano pratico). Con un numero di fasi minori, per esempio 8, il costo si riduce drasticamente e allora il costo è minore rispetto alla ricerca esaustiva.

14.5.3 Crittoanalisi lineare

Nel 1993 sono stati scoperti attacchi di *crittoanalisi lineare* che consistono nella costruzione di una approssimazione lineare della S-box che permette di inferire taluni bit della chiave, il resto si *bruta*.

Costo Si scende a 2^{43} coppie $\langle m, c \rangle$ ed è di tipo known plain-text. Metodo più efficiente del forza bruta.

14.6 Varianti del DES

Il DES è sensibile agli attacchi di crittoanalisi differenziale e lineare, per renderlo più robusto si sono introdotte delle varianti.

14.6.1 Scelta indipendente delle sottochiavi

Anziché generare le sottochiavi di fase a partire dalla chiave principale si scelgono manualmente. E' come passare da 56 bit a $16 \cdot 48 = 768$ bit di chiave.

Problema risolto? Non abbiamo un effettivo aumento della sicurezza perché è sempre vulnerabile ad attacchi differenziali: si passa da 768 bit di sicurezza a 61 bit (2^{61} la complessità).

14.6.2 Cifratura multipla: 2DES

Questa variante ha avuto maggiore successo rispetto alla precedente. L'idea è di applicare due volte la cifratura, di comporre il DES con se stesso:

$$\forall k_1, k_2, k_3 : C_D(C_D(m, k_1), k_2) \neq C_D(m, k_3)$$

applicare due volte la cifratura DES non è uguale ad applicarlo una sola volta. Si arriva ad uno spazio delle chiavi di 2^{112} .

Problema risolto? L'attacco *meet-in-the-middle* fa scendere la sicurezza a 57 bit di sicurezza:

$$c = C(C(m, k_1), k_2)$$

facciamo la decifrazione di C con la chiave k_2 (tolgo la cifratura più esterna)

$$D(c, k_2) = C(m, k_1)$$

possiamo quindi prendere una coppia $\langle m, c \rangle$:

- $\forall k_1$ calcolo e salvo $C(m, k_1) \rightarrow 2^{56}$ cifrature (tutte)
- $\forall k_2$ calcolo e verifico corrispondenza di $D(c, k_2) \rightarrow 2^{56}$ cifrature (al più)

Cercando nella lista delle cifrature, so che esisterà una coppia $< k_1, k_2 >$ per la quale c'è la corrispondenza:

$$D(c, \bar{k}_2) = C(m, \bar{k}_1)$$

Questo attacco costa quindi $2^{56} + 2^{56} = 2 \cdot 2^{56} = 2^{57}$ al più, tra cifrature e decifrature. Cioè l'aumento di complessità dovuto alla doppia cifratura non è di N^2 ma di $2N$

$$2N << N^2$$

14.6.3 Cifratura multipla: 3DES

Nella pratica non si è fatta una doppia cifratura.

- **2TDEA** (proposta a due chiavi):

$$c = C(D(C(m, k_1), k_2), k_1)$$

k_1 e k_2 sono chiavi di 56 bit tra di loro indipendenti. Si sceglie questa modalità (CDC invece di CCC) per mantenere la compatibilità abilitati a usare un singolo DES. Se pongo $k_1 = k_2$ ottengo il DES normale (cifratura singola).

Attenzione: CDC non è più robusto di CCC. Un attacco *meet-in-the-middle* ha un costo di 112 bit.

- **3TDEA** (*triple data encryption algorithm*):

$$c = C(D(C(m, k_1), k_2), k_3)$$

k_1, k_2, k_3 sono chiavi a 56 bit. Si hanno chiavi di $3 \cdot 56 = 168$ bit. In realtà è vulnerabile ad attacchi *meet-in-the-middle* quindi la sicurezza scende da 168 bit a 112 bit di sicurezza.

Per ottenere il messaggio decifriamo usando le chiavi in ordine inverso

$$m = D(C(D(c, k_3), k_2), k_1)$$

Osserviamo banalmente che

$$C(m, k_1) = C(D(c, k_3), k_2)$$

come prima

$$\forall k_1 \text{ calcolo e salvo } C(m, k_1)$$

successivamente

$$\forall k_2, k_3 \text{ calcolo } C(D(c, k_3), k_2) \text{ e lo cerco nella lista}$$

A livello di costo otteniamo $O(2^{56} + 2^{112}) = O(2^{112})$.

Capitolo 15

Advanced Encryption Standard (AES)

15.1 Storia

Nuovo bando Nel giugno 1998 esce il bando organizzato dal NIST per individuare un nuovo algoritmo in grado di soppiancare il DES ed il 3DES. Le proposte dovevano tenere conto dei seguenti elementi:

- **sicurezza**: resistere a tutti gli attacchi noti
- **costo di realizzazione**: doveva essere facile implementarlo sia in software che in hardware
- **libero da brevetti** in quanto il nuovo standard doveva divenire libero da utilizzare
- **caratteristiche algoritmiche**: portabile su diverse macchine, usabile con chiavi di diversa lunghezza [Novità!]

Ci furono 21 proposte. Già nell'agosto 1998 furono scartati 6 cifrari, e nell'aprile 1999 ne rimasero solo 5.

Sfida finale I cifrari arrivati alla fine furono:

- MARS (IBM)
- RC6 (RSA)
- Rijndael (Proton word int + Università di Leuven, Belgio)
- SERPENT (Università di Israele, UK, USA)
- TWOFLASH (Berkeley, Princeton)

Scelta definitiva In ottobre 2000 Rijndael vinse il bando e diventò AES con qualche modifica. Nel 2001 AES diventa ufficialmente lo standard. E' un algoritmo che ancora oggi continua a conservare tutti i bit di sicurezza.

15.2 Caratteristiche

15.2.1 Dimensione della chiave e numero di fasi

L'AES accetta chiavi da 128, 192, 256 bit. Noi vedremo la versione a 128 bit (ancora oggi tutti bit di sicurezza). E' anch'esso un cifrario a fasi (come il DES), dove il numero di fasi dipende dal numero di bit della chiave:

- 10 fasi → 128 bit
- 12 fasi → 192 bit
- 14 fasi → 256 bit

15.2.2 Preparazione: selezione delle sottochiave di ogni fase

Possibile scegliere in anticipo tutte le chiavi.

La prima differenza rispetto al DES è il processo di selezione delle sottochiavi di fase. Ogni fase utilizza 128 bit di chiave, a partire da questa calcoliamo nuove chiavi attraverso un processo deterministico. Collochiamo la nostra chiave in una matrice 4×4 (ogni elemento contiene 8 bit, 32 bit per colonna):

$$K = \begin{bmatrix} K[0] & K[4] & K[8] & K[12] \\ K[1] & K[5] & K[9] & K[13] \\ K[2] & K[6] & K[10] & K[14] \\ K[3] & K[7] & K[11] & K[15] \end{bmatrix}$$

Nelle dieci fasi successive diamo in input nuove matrici K . La cosa nuova è che il procedimento utilizzato nelle fasi per ricavare le chiavi non è lineare (come invece è nel DES).

Procedimento Prese le colonne in ordine da sinistra verso destra abbiamo:

$$W(0), W(1), W(2), W(3)$$

Costruiamo quindi $W(i)$ che è una sequenza di byte che usiamo per generare le sottochiavi di fase. Per ogni $t \geq 4$ otteniamo

$$W(t) = \begin{cases} W(t-1) \oplus W(t-4) & \text{se } t \text{ non è multiplo di 4} \\ T(W(t-1)) \oplus W(t-4) & \text{se } t \text{ è multiplo di 4} \end{cases}$$

T è una funzione non lineare applicata tramite una S-box.

Chiave dell' i -esima fase La chiave dell' i -esima fase è quindi composta da:

$$W(4 \cdot i), W(4 \cdot i + 1), W(4 \cdot i + 2), W(4 \cdot i + 3)$$

Complessivamente abbiamo fatto un'espansione non lineare da 4 colonne a 44 colonne. Le chiavi per tutte le fasi sono pronte.

15.2.3 Preparazione: cifratura per blocchi

La cifratura si fa per blocchi (di 128 bit in questo caso). Si creano i blocchi caricandoli *per colonna* in una matrice 4×4 (anche in questo caso abbiamo 1 byte in ogni elemento):

$$B = \begin{bmatrix} b_{00} & b_{01} & b_{02} & b_{03} \\ b_{10} & b_{11} & b_{12} & b_{13} \\ b_{20} & b_{21} & b_{22} & b_{23} \\ b_{30} & b_{31} & b_{32} & b_{33} \end{bmatrix} \quad b_{ij} \in \{0, 1\}^8$$

Prima trasformazione Si applica una prima trasformazione applicando la chiave:

$$B \longrightarrow B \oplus K$$

15.2.4 Fasi

Domanda da esame. Descrivere il cifrario AES, e in particolare le quattro operazioni eseguite in ciascuna fase.

Dopodiché si inizia con le fasi, ognuna di esse composta da 4 operazioni:

- *substitute bytes* (applicazione della S-box)
- *shift rows* (dipendenza degli elementi di una colonna dagli elementi delle altre colonne)
- *mix columns* (non si applica nella decima fase, dipendenza dell'elemento di una colonna dagli altri elementi della colonna)
- *add round key* (applicazione della chiave i -esima)

Le prime 3 operazioni applicano: non linearità, diffusione e confusione. L'ultima fase aggiunge la chiave. Alla fine delle iterazioni si ottiene il crittogramma. La S-Box dell'AES è sempre usata in forma di tabella di verità, ma a differenza di quella del DES si conosce la funzione che la genera!

15.2.4.1 Substitute bytes (S-box)

Ogni byte viene trasformato usando la S-Box:

$$b_{ij} \rightarrow \text{S-Box}(b_{ij})$$

La S-Box è diversa rispetto a quella del DES, dove la S-box è stata imposta in sola forma tabellare (**uno degli aspetti più criticati del DES**, principalmente per sospetti di *trapdoor*). Questa S-Box è presentata in forma tabellare, ma si conosce in modo chiaro le operazioni algebriche eseguite: si compone di una matrice 16×16 di interi $\in [0, 255]$

e contiene una permutazione dei numeri di questo intervallo. L'accesso alla S-Box viene fatto suddividendo il byte b_{ij} in due blocchi da 4 bit l'uno:

$$b_{ij} = b_1 b_2 b_3 b_4 \mid b_5 b_6 b_7 b_8$$

i primi 4 bit formano il numero di *riga* ($0 \leq x \leq 15$), gli ultimi 4 bit formano il numero di *colonna* ($0 \leq y \leq 15$).

Esempio Prendiamo

$$b_{ij} = 10001011$$

spezziamo a metà: 1000 è 8, mentre 1011 è 11. Nella tabella che definisce la S-box otterremo

$$\text{S-box}[8, 11] = 61 = (00111101)_2$$

Relazione algebrica implementata La relazione implementata dalla S-Box è:

$$x \rightarrow x^{-1} + c$$

questo inverso è l'*inverso moltiplicativo del byte* (spiegato più avanti nelle nozioni di Algebra lineare) calcolato nel campo $\text{GF}(2^8)$ (*Campo di Galois*, campo finito) con l'aggiunta di una componente lineare. In questo insieme:

- la somma viene eseguita tramite lo XOR (somma modulo 8)
- la moltiplicazione è eseguita mod 2^8 .

Nella pratica ogni byte può essere visto come un polinomio, facendo la moltiplicazione si fa il prodotto tra polinomi ma in modulo, quindi si taglano via i gradi oltre al settimo.

Miglioramento Questa S-Box ha una bassissima correlazione tra i bit di ingresso e di uscita. Inoltre essendo usata sia per la chiave che per la matrice del messaggio si ha una maggiore sicurezza

15.2.4.2 *Shift rows* (Diffusione)

Si usa per spargere il messaggio. Lascia invariata la prima riga, shifta a sx le altre righe, rispettivamente di 1, 2 e 3 posizioni.

$$\begin{bmatrix} b_{00} & b_{01} & b_{02} & b_{03} \\ b_{10} & b_{11} & b_{12} & b_{13} \\ b_{20} & b_{21} & b_{22} & b_{23} \\ b_{30} & b_{31} & b_{32} & b_{33} \end{bmatrix} \rightarrow \begin{bmatrix} b_{00} & b_{01} & b_{02} & b_{03} \\ b_{11} & b_{12} & b_{13} & \mathbf{b}_{10} \\ b_{22} & b_{23} & \mathbf{b}_{20} & \mathbf{b}_{21} \\ b_{33} & \mathbf{b}_{30} & \mathbf{b}_{31} & \mathbf{b}_{32} \end{bmatrix}$$

Ho quindi diffuso i byte di ogni colonna su tutte le altre.

15.2.4.3 Mix columns (Diffusione)

Si prende ogni colonna della matrice, viste come vettori, e si moltiplicano per una matrice. Abbiamo una matrice M di dimensione 4×4 byte, per ogni colonna:

$$B_j \rightarrow M \cdot B_j$$

con $0 \leq j \leq 3$. Anche qui si lavora in $\text{GF}(2^8)$. **Questo porta ogni elemento della colonna ad essere dipendente da tutti i byte della colonna.** La matrice M è pensata in modo tale che ogni byte di una colonna influenzi tutti i byte della stessa colonna.

b_{ij} dipende da tutti i byte: $b_{0j}, b_{1j}, b_{2j}, b_{3j}$

15.2.4.4 Add round key (Confusione)

Facciamo somma della chiave di base, poco da dire:

$$b_{ij} \rightarrow b_{ij} \oplus K_{ij} \text{ chiave della sottofase}$$

15.2.5 Cipher Block Chaining (CBC)

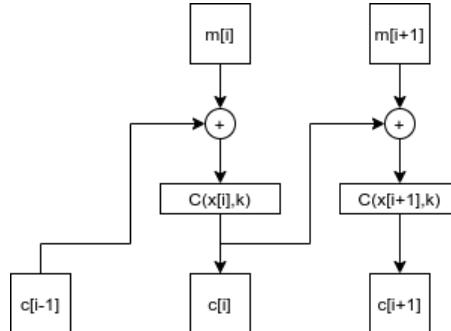
Sia il DES che l'AES cifrano il *plaintext* a blocchi, AES ad esempio divide in blocchi grandi quanto la chiave e poi si critpa. Il problema di ciò è che blocchi uguali vengono cifrati allo stesso modo (si usa sempre la stessa chiave in una sessione), questo può dare molte informazioni al crittoanalista.

Soluzione Per risolvere questo problema si stabilisce una dipendenza tra blocco i -esimo e precedenti. Nell'AES si prende il messaggio m e lo si divide in blocchi da 128 bit:

$$m = m_1 m_2 \dots m_i \dots m_e$$

- se $|m_e| < 128$ allora si aggiunge una sequenza 10...0 fino ad arrivare a 128 bit;
- se invece $|m_e| = 128$ si inserisce comunque un intero blocco 10...0 in modo da avere sempre questo terminatore.

Successivamente si sceglie una sequenza iniziale c_0 random, che può anche essere trasmessa in chiaro, e si attua questo schema:

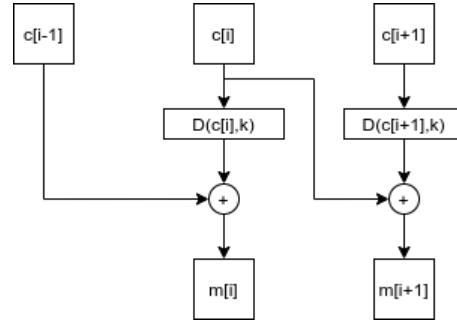


$$x_i = m_i \oplus c_{i-1}$$

$$c_i = C(m_i \oplus c_{i-1}, K)$$

questa modalità è detta *Cipher Block Chaining* (CBC).

Decifrazione La decifrazione invece si esegue in questo modo:



$$m_i = D(c_i, K) \oplus c_{i-1}$$

Osservazione Mentre la cifratura va eseguita necessariamente sequenzialmente (visto che un elemento dipende dal precedente) la decifrazione si può eseguire in parallelo (abbiamo tutti gli elementi, non dobbiamo attendere il precedente). Inoltre se invio il testo cifrato e ci sono errori risulta in un errata decifrazione solo del blocco i e $i + 1$ mentre gli altri continuano ad essere decriptati correttamente.

Parte VI

Crittografia a chiave pubblica

Capitolo 16

Introduzione

16.1 Problema base: scambio delle chiavi su canali insicuri

Abbiamo visto che il One-Time Pad è assolutamente sicuro, ma:

- richiede una chiave segreta nuova per ogni messaggio;
- la chiave deve essere perfettamente casuale e lunga come il messaggio da cifrare.

AES ha chiavi più corte, ma necessita comunque di uno scambio di chiavi. Segue la domanda: come si può scambiare una chiave in maniera sicura? La cosa può essere risolta in due modi:

- protocollo Diffie-Hellman;
- crittografia a chiave pubblica.

Protocollo Diffie-Hellman (DH) Nel 1976 Diffie ed Hellman con il loro articolo *New Directions in Cryptography* hanno rivoluzionato la crittografia. Il protocollo *DH* (Diffie-Hellman) permette la negoziazione di una chiave di sessione senza che le due parti si siano scambiate informazioni in precedenza.

16.2 Schema della crittografia a chiave pubblica

In parallelo sempre gli stessi Diffie ed Hellman propongono lo schema di crittografia a chiave pubblica, senza tuttavia averne una valida implementazione.

Chiavi In questo nuovo schema si hanno due chiavi:

- una *pubblica* nota a tutti ed utilizzata per cifrare
- una *privata* nota solo al ricevente usata per decifrare

Ogni utente avrà quindi una coppia $\langle K_{priv}, K_{pub} \rangle$: l'utente rende nota la chiave pubblica e mantiene riservata quella privata. Abbiamo $2N$ chiavi in totale (dove N è il numero di utenti, in contrasto con le N^2 chiavi richieste in un cifrario simmetrico).

Funzione di cifratura e decifrazione La cifratura ha la forma:

$$c = C(m, K_{pub})$$

mentre la decifrazione ha la forma:

$$m = D(c, K_{priv})$$

Entrambe le funzioni sono note al pubblico, così come le chiave pubblica. La chiave privata è nota al solo destinatario, dunque solo il destinatario è in grado di decifrare il messaggio.

Asimmetricità Il cifrario si dice asimmetrico perché le due parti hanno ruoli diversi, in particolare non c'è più la condivisione di un segreto comune.

16.2.1 Caratteristiche del cifrario

Questi cifrari devono avere alcune caratteristiche:

- **il procedimento di cifratura e decifrazione deve essere corretto**, il destinatario deve poter risalire al messaggio in chiaro

$$D(C(m, K_{pub}), K_{priv}) = m$$

- **il sistema deve essere efficiente e sicuro**, cioè

- la coppia di chiavi deve essere facile da generare, e deve risultare praticamente impossibile che due utenti scelgano la stessa chiave pubblica (altrimenti la chiave privata risulterebbe la stessa);
- dato un messaggio m e la chiave pubblica K_{pub} deve essere facile per il mittente calcolare il crittogramma
- dato un crittogramma c e la chiave privata K_{priv} deve essere facile per il destinatario calcolare il messaggio originale
- pur conoscendo il crittogramma c , la chiave pubblica K_{pub} e le funzioni di cifratura e decifrazione, deve essere difficile per il crittoanalista risalire a m .

I due requisiti elencati possono essere soddisfatti ricorrendo alle *funzioni one-way trap-door*, funzioni facili da calcolare ma **difficili da invertire** senza avere maggiori informazioni.

RSA Nel 1977 Rivest, Shamir ed Adleman scoprono una funzione di questo tipo basata sui numeri primi. L'algoritmo RSA (*Rivest-Shamir-Adleman*) si basa sul fatto che

- dati due numeri primi p e q calcolare il prodotto $n = p \cdot q$, mentre
- dato n è difficile fare la fattorizzazione per trovare p e q (a meno che non si conosca uno dei due fattori).

Prima di introdurre l'RSA è necessario introdurre nozioni di algebra modulare.

16.2.2 Vantaggi e svantaggi

- **Pro:**

- meno chiavi
- **non richiedono lo scambio di chiavi** (si risolve problema tipico della crittografia simmetrica)

- **Con:**

- più lenti della cifratura simmetrica
- esposizione ad attacchi *chosen plain-text*

Si può infatti criptare un tot di messaggi m_1, \dots, m_h e vedere se sul canale passa uno dei vari c_1, \dots, c_h . Se compare allora sappiamo il messaggio, se non compare allora sappiamo quale messaggio sicuramente non è.

Adotteremo un ibrido tra crittografia a chiave pubblica e cifrari simmetrici.

Si legga le nozioni di Algebra modulare in fondo alla dispensa prima di proseguire.

16.3 Generatori

Sia data la seguente funzione

$$a^k \bmod n$$

con $a \in Z_n^*$ e $1 \leq k \leq \phi(n)$. Si dice che a è un *generatore di Z_n^** se la funzione genera **tutti e soli** gli elementi di Z_n^* .

Ordine degli elementi L'ordine degli elementi è difficile da prevedere. L'unica cosa prevedibile è che 1 si ottiene **solo** quando

$$x = \phi(n)$$

poiché sappiamo $a^{\phi(n)} \equiv 1 \pmod{n}$ per il teorema di Eulero.

Esempi

- 2 è generatore di Z_{13}^*

$$Z_{13}^* = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\} \quad \phi(13) = 12$$

$$2^k \bmod 13 = 2, 4, 8, 3, 6, 12, 11, 9, 5, 10, 7, 1 \quad 1 \leq k \leq 13$$

- 3 è generatore di Z_7^*

$$Z_7^* = \{1, 2, 3, 4, 5, 6\} \quad \phi(7) = 6$$

$$3^k \bmod 7 = 3, 2, 6, 4, 5, 1 \quad 1 \leq k \leq 6$$

- 2 non genera Z_7^*

$$2^3 \bmod 7 = 2^6 \bmod 7 = 1$$

facendo le potenze con modulo otteniamo 1 generato per valori diversi da $\phi(n)$, e questo non va bene!

16.3.1 Teorema sull'esistenza di almeno un generatore

Se n è primo allora Z_n^* ha almeno un generatore.

- **Non tutti gli elementi sono generatori.**

Per n primo non tutti gli elementi di Z_n^* sono suoi generatori (1 non lo è mai).

- **Numero di generatori.** Per n primo i generatori di Z_n^* sono $\phi(n - 1)$.

Numeri non primi Non tutti gli n forniscono generatori per Z_n^* , ad esempio Z_8^* .

16.3.2 Problema: individuare i generatori

Determinare un generatore di Z_n^* con n primo è difficile, si possono provare tutti i numeri in $[2, n - 1]$ ma è esponenziale nella dimensione di n . Esistono tuttavia algoritmi randomizzati che risolvono il problema con altissima probabilità di successo.

16.3.2.1 Problema del logaritmo discreto

Il calcolo consiste nel trovare x che risolve:

$$a^x \equiv b \pmod{n}, n \text{ primo}$$

L'equazione ammette soluzioni $\forall b$ se e solo se a è un generatore di Z_n^* . Tuttavia non possiamo sapere in che ordine sono generati i valori di Z_n^* quindi per trovare x giusto dobbiamo iterare tutti i valori.

Rendiamo chiara l'idea L'algebra modulare complica le cose. Prendiamo la potenza 2^x : nell'algebra modulare si perde struttura, e quindi la possibilità di costruire algoritmi efficienti.

x	1	2	3	4	5	6	7	8	9	10	11	12
2^x	2	4	8	16	32	64	128	256	512	1024	2048	4096
$2^x \pmod{13}$	2	4	8	3	6	12	11	9	5	10	7	1

- **Esempio dall'algebra NON modulare.**

Sappiamo che

$$2^x = 512$$

e vogliamo trovare x . Supponiamo che la soluzione sia $x = 8$: facendo i calcoli ci rendiamo conto di avere sbagliato. La cosa non è vana perché otteniamo informazioni: osservo che $2^8 = 256$, quindi è stato scelto un numero x troppo basso (escluso i valori minori di otto). Provo i numeri maggiori: con $x = 9$ otteniamo 512!

- **Esempio nell'algebra modulare.**

Sappiamo che

$$2^x \pmod{13} = 5$$

e vogliamo trovare x . L'assenza di struttura ordinata mi impedisce di ottenere informazioni dopo aver compiuto errori. Poniamo $x = 6$

$$2^6 \bmod 13 = 12$$

il risultato non è giusto, ma contrariamente a prima non posso escludere i valori di x minori o quelli superiori: devo provarli per forza tutti! A un certo punto troveremo che la soluzione è $x = 9$.

16.4 Funzioni *one-way trap-door*

Le funzioni *one-way trap-door* permettono la ostruzione di cifrari a chiave pubblica:

- *one-way* in quanto funzioni facili da calcolare, ma difficili da invertire
- *trap-door* per la possibilità di calcolare l'inverso in presenza di alcune informazioni

16.4.1 Esempio: problema della fattorizzazione

Dati p, q primi è facile fare il prodotto

$$n = p \cdot q$$

Non possiamo dire la stessa cosa se dato n vogliamo trovare p e q : è necessario tempo (sub)esponenziale.

Trapdoor Dato p o q , invece, si fattorizza velocemente: $q = \frac{n}{p}$, ma anche conoscendo $\phi(n)$ si fattorizza velocemente. Lo si può fare in $O(\log_2 z)$.

RSA Questo problema determina la "robustezza" del cifrario RSA.

16.4.2 Esempio: calcolo della radice in modulo

Abbiamo visto che è sufficiente tempo polinomiale per trovare y (algoritmo delle quadrate successive)

$$y = x^z \bmod s$$

Se s non è primo, tuttavia, calcolare la base x (distinguere dal problema del logaritmo discreto, dove si va a calcolare l'esponente e non la base) richiede tempo esponenziale

$$x = y^{\frac{1}{z}} \bmod s$$

Applicazioni Tutte le volte che chiediamo due interi primi p e q andiamo a calcolare un valore n non primo. n

Attenzione Parliamo del calcolo della radice in modulo, non del calcolo della radice e basta! Il calcolo della radice senza modulo è facile da calcolare (si osservi che nell'RSA si deve stare attenti a non avere $m < n$ con $m \bmod n$)

Capitolo 17

RSA (Rivest-Shamir-Adleman)

17.1 Generazione delle chiavi

La creazione della coppia di chiavi spetta al destinatario.

1. Si scelgono due numeri interi e primi p, q molto grandi (≈ 1000 bit) (adesso si consiglia che $p \cdot q \approx 2048$ bit per avere una protezione fino al 2030, mentre per una protezione di maggiore durata si consigliano 3072 bit). Ovviamente questo passo deve essere fatto in tempo polinomiale (la generazione di p e q): si generano sequenze casuali (nei modi già visti) e si testa la primalità utilizzando il test di Miller-Rabin.
2. Si calcola il prodotto n e la funzione di Eulero $\phi(n)$ (questo è facile da calcolare, poichè conosciamo la scomposizione di n , si ricordi la formula vista indietro)

$$\begin{aligned} n &= p \cdot q \\ \phi(n) &= (p - 1) \cdot (q - 1) \end{aligned}$$

Anche qui operazioni in tempo polinomiale nella grandezza dell'input.

3. Si sceglie $e < \phi(n)$ tale che e e $\phi(n)$ siano coprimi

$$\text{MCD}(e, \phi(n)) = 1$$

4. Si calcola

$$d = e^{-1} \bmod \phi(n)$$

Anche questo si può calcolare in tempo polinomiale usando l'algoritmo di Euclide esteso. Si osservi che essendo $MCD(e, \phi(n)) = 1$ si ha unicità della soluzione (e^{-1}).

Risultato finale Alla fine si hanno quindi le due chiavi:

$$K_{pub} = \langle e, n \rangle$$

$$K_{priv} = \langle d, n \rangle$$

Naturalmente va tenuto segreto anche il resto delle informazioni: $p, q, \phi(n)$.

17.2 Cifratura e decifratura

I messaggi sono sequenze binarie trattate come interi!

- **NB.** Indichiamo il messaggio m e imponiamo $m < n$. In caso contrario potrebbero emergere ambiguità nella decifrazione: la cosa è chiara nella dimostrazione di correttezza, non sarà possibile arrivare a conclusione senza adottare questa condizione. Se $m \geq n$ allora dividiamo il messaggio m in blocchi di $b = \lfloor \log_2 n \rfloor$ bit (sempre con la stessa chiave) e cifriamo ciascuno (eventualmente con CBC).
- **NB2.** Solitamente si fissa un limite inferiore comune per la dimensione dei blocchi, prevenendo così l'uso di cifrari poco sicuri. Supponiamo che il blocco sia di almeno b bit: i messaggi saranno numeri fino a 2^b , ma lo stesso massimo deve essere minore di n per la condizione detta

$$m < 2^b < n$$

- **Cifratura.** Abbiamo la chiave pubblica $\langle e, n \rangle$ e il messaggio m

$$c = C(m, K_{pub}) = m^e \bmod n$$

con $m < n, c < n$

- **Decifrazione.** Abbiamo la chiave privata $\langle d, n \rangle$ e il crittogramma c

$$m = D(c, K_{priv}) = c^d \bmod n$$

Sicuramente il cifrario RSA è molto elegante! L'utilizzo del modulo garantisce la sicurezza del cifrario.

17.2.1 Esempio

Consideriamo il seguente esempio.

1. Calcoliamo p, q e la funzione di Eulero

$$\begin{aligned} p = 5, q = 11 \implies n &= p \cdot q = 5 \cdot 11 = 55 \\ \phi(n) &= (p - 1) \cdot (q - 1) = 40 \end{aligned}$$

2. Scegliamo un valore e coprimo con 40: prendiamo $e = 7$, che è coprimo con $\phi(n) = 40$ poichè

$$\text{MCD}(7, 40) = 1$$

3. Applichiamo l'algoritmo di Euclide esteso per calcolare l'inverso moltiplicativo $d = 7^{-1} \bmod 40$

```

EE(7,40) ---> <1, -17>
EE(40, 7 mod 40) = EE(40, 7) ---> <1,3,-2 - floor(40/7)*3 = <1,3,-17>
EE(7, 40 mod 7) = EE(7, 5) ---> <1, -2, 1 - floor(7/5)*(-2)> = <1,-2,+3>
EE(5, 7 mod 5) = EE(5, 2) ---> <1,1,0 - floor(5/2)*1> = <1,1,-2>
EE(2, 5 mod 2) = EE(2, 1) ---> <1,0, 1 - floor(2/1)*0> = <1,0,1>
EE(1, 0) ---> <1,1,0>

```

L'inverso risultante è -17 , che è negativo: visto che lavoriamo in modulo 40 sottraiamo 40. Morale della favola:

$$d = -17 \bmod 40 = 23 \bmod 40 = 23$$

4. Abbiamo ottenuto

$$K_{pub} = <7, 55> \quad K_{priv} = <23, 55>$$

Per decifrare faremo ($m < 55$)

$$\begin{aligned} c &= m^7 \bmod 55 \\ m &= c^{23} \bmod 55 \end{aligned}$$

Si possono eseguire i calcoli in tempo polinomiale tramite l'algoritmo delle quadrate successive.

17.3 Dimostrazione del teorema di correttezza del cifrario

Dimostrare la correttezza del cifrario significa dimostrare che

$$D(C(m, K_{pub}), K_{priv}) = m$$

nel contesto del RSA significa dimostrare che ($\forall m < n$)

$$c^d \bmod n = (m^e \bmod n)^d \bmod n = \boxed{m^{ed} \bmod n = m}$$

La cosa non è così ovvia: e e d sono l'uno l'inverso dell'altro ma rispetto a mod $\phi(n)$ e non mod n (**ricordare come abbiamo generato la chiave**)

$$ed \equiv 1 \bmod \phi(n)$$

La dimostrazione del teorema di correttezza del cifrario si fa affrontando tre casi. Si prenda il prodotto $n = p \cdot q$ e chiediamoci: cosa succede se provo a dividere il messaggio m (che trattiamo come intero) per q ? Cosa succede se faccio lo stesso con p ?

17.3.1 Caso: il messaggio m è divisibile sia per q che per p

Avere m divisibile sia per p che per q significa che è divisibile per n .

Ciò non è possibile perché abbiamo imposto $m < n$.

La divisione risulta possibile solo se m è almeno uguale ad n (e quindi se $m \geq n$). ■

17.3.2 Caso: p e q non dividono il messaggio m

Se p e q non dividono m allora m ed n sono coprimi, cioè: $\text{MCD}(m, n) = 1$. Riprendiamo le seguenti cose.

- **Teorema di Eulero:** $m^{\phi(n)} \equiv 1 \pmod{n}$ (possiamo dirlo visto che $\text{MCD}(m, n) = 1$)
- **Definizione di inverso** (con $r \in \mathbb{N}$): $ed \equiv 1 \pmod{\phi(n)} \implies ed = 1 + r \cdot \phi(n)$

Partendo da quest'ultima cosa scriviamo

$$m^{ed} \pmod{n} = m^{1+r \cdot \phi(n)} \pmod{n} = m \cdot (m^{\phi(n)})^r \pmod{n}$$

Abbiamo detto che m ed n sono coprimi, quindi possiamo utilizzare il teorema di Eulero.

$$m \cdot 1^r \pmod{n} = m \pmod{n} = m$$

L'ultimo passaggio è certo (la rimozione del modulo) poichè $m < n$ ■

17.3.3 Caso: il messaggio m è divisibile per p e non per q (o viceversa)

Supponiamo che m e n non siano coprimi: in particolare abbiamo m divisibile per p e non per q (o viceversa). Non possiamo usare il teorema di Eulero come appena fatto!

Numero divisibile Se p è multiplo di m allora

$$m \equiv 0 \pmod{p} \implies m \pmod{p} = 0$$

tutte le potenze di m continueranno ad essere divisibili per p , quindi

$$\forall r \in \mathbb{N} \quad m^r \equiv 0 \pmod{p} \implies m^r \pmod{p} = 0$$

siccome sia m^r che m sono congrui a 0 modulo p allora lo è pure la differenza

$$\forall r \in \mathbb{N} \quad m^r - m \equiv 0 \pmod{p} \implies (m^r - m) \pmod{p} = 0$$

Possiamo porre $r = ed$

$$m^{ed} - m \equiv 0 \pmod{p}$$

Numero non divisibile Ricaviamo una relazione simile anche per q . Sappiamo che q ed m sono coprimi, dunque è applicabile il teorema di Eulero

$$m^{\phi(q)} \equiv 1 \pmod{q} \implies m^{\phi(q)} \pmod{q} = 1$$

Facciamo i seguenti calcoli:

$$\begin{aligned} m^{ed} \pmod{q} &= m^{\phi(n) \cdot r + 1} \pmod{q} = m \cdot m^{r(p-1) \cdot (q-1)} \pmod{q} = m \cdot (m^{(q-1)})^{(p-1) \cdot r} \pmod{q} = \\ &= m \cdot (m^{\phi(q)})^{(p-1) \cdot r} \pmod{q} = m \pmod{q} \end{aligned}$$

q è numero primo, dunque la funzione di Eulero è il numero decrementato ($\phi(q) = q - 1$). Concludiamo:

$$m^{ed} \equiv m \pmod{q} \implies m^{ed} - m \equiv 0 \pmod{q}$$

Conclusione Abbiamo quindi ottenuto:

$$\begin{cases} m^{ed} - m \equiv 0 \pmod{q} \\ m^{ed} - m \equiv 0 \pmod{p} \end{cases}$$

che significa che $m^{ed} - m$ è divisibile sia per p che per q quindi lo sarà anche per $n = p \cdot q$, possiamo quindi dire:

$$m^{ed} - m \equiv 0 \pmod{n} \implies m^{ed} \equiv m \pmod{n} = m$$

L'ultimo passaggio è certo (la rimozione del modulo) poiché $m < n$

■

17.4 Sicurezza del cifrario ed attacchi

17.4.1 Soluzioni equivalenti alla fattorizzazione

La sicurezza è legata alla difficoltà della fattorizzazione di un numero molto grande.

La fattorizzazione è C.S. (non si sa se C.N.) per rompere RSA in tempo polinomiale

Se fattorizzo n trovo p e q , e quindi $\phi(n) = (p-1)(q-1)$. Consideriamo le possibili alternative, tutte equivalenti alla fattorizzazione.

- **Calcolo della radice in modulo.**

$$m = \sqrt[e]{c} \pmod{n}$$

Il calcolo è difficile **almeno** quanto la fattorizzazione di n (n composto), non emerge utilità nello svolgere questi calcoli.

- **Calcolo di $\phi(n)$ direttamente da n .**

Il calcolo di $\phi(n)$ a partire da n è computazionalmente equivalente alla fattorizzazione di n : se io calcolo $\phi(n)$ da n allora conoscendo i due posso trovare p e q in tempo polinomiale.

$$n = p \cdot q \rightarrow \phi(n) = (p-1) \cdot (q-1)$$

Calcoliamo $\phi(n)$

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

ottengo $x_1 = p+q = n - \phi(n) + 1$, poi $(p-q)^2 = (p+q)^2 - 4 \cdot n = (x_1)^2 - 4 \cdot n$

quindi $x_2 = p-q = \sqrt{x_1^2 - 4 \cdot n}$

ottengo che i due problemi sono equivalenti.

$$\begin{cases} x_1 = p+q \\ x_2 = p-q \end{cases} \implies \begin{cases} p = \frac{x_1+x_2}{2} \\ q = \frac{x_1-x_2}{2} \end{cases}$$

- **Attacco esauriente su d .**

Un altro attacco che posso fare è cercare di brutare d partendo da (n, e) .

NB Fattorizzare non è *NP-hard* quindi non si esclude che esista un algoritmo efficiente.

17.4.2 Osservazioni sulla fattorizzazione di n

La fattorizzazione è difficile, ma non più come una volta: l'hardware migliora e gli algoritmi si affinano (pur rimanendo costosi sul piano computazionale). Non è definito un limite inferiore: questo significa che in futuro potrebbero emergere algoritmi aventi migliore complessità.

- Esistono algoritmi di **complessità sub-esponenziale** come il *General Number Field Sieve* (GNFS) che richiede $O(2^{\sqrt{b} \cdot \log b})$ con b la dimensione di n (un attacco brute-force è $O(2^b)$).
- Con la potenza di calcolo attuale usando GNFS si fattorizza fino a 768 bit. Nuovi algoritmi hanno ridotto il numero di bit di sicurezza, ergo hanno comportato un aumento del numero di bit necessari per garantire la sicurezza. Per avere una sicurezza come quella che si ottiene con AES a 128 bit si dovrebbe ricorrere a n con circa 3072 bit, si sale addirittura a 15360 bit di modulo se lo si compara con AES256.

TDEA, AES (bit della chiave)	RSA e DH (bit del modulo)
80	1024
112	2048
128	3072
192	7680
256	15360

Nell'AES i bit della chiave sono tutti bit di sicurezza, mentre nel RSA i bit del modulo non sono tutti di sicurezza: **si hanno attacchi più efficienti del forza bruta** (motivo per cui nel tempo i bit aumentano).

17.4.3 Scelta ottimale dei valori p, q

I numeri p e q vanno scelti *molto* grandi, *entrambi* attorno ai 1700 bit: se uno è troppo piccolo la fattorizzazione ci mette poco a individuare il numero piccolo.

- Sia $p - 1$ che $q - 1$ devono contenere fattori grandi (altrimenti n si fattorizza velocemente).
- $\text{MCD}(p-1, q-1)$ deve essere piccolo! Conviene scegliere p e q tali che i numeri $\frac{p-1}{2}$ e $\frac{q-1}{2}$ siano coprimi.
- **Norma di buon senso: cambiare sempre entrambi i numeri primi.**

Mai riutilizzare uno dei primi per altri moduli: il crittoanalista potrebbe pensare che io per pigrizia ho generato un nuovo numero primo mantenendo l'altro numero primo. La cosa non è un problema, sappiamo ottenere i numeri primi in tempo polinomiale.

$$\begin{cases} n_1 = p \cdot q_1 \\ n_2 = p \cdot q_2 \end{cases} \implies \text{MCD}(n_1, n_2) = p$$

- **Attacco con p e q simili.**

Scegliere p e q distanti tra loro! Se $p \approx q$ allora si ha un attacco molto efficiente dove si ipotizza

$$n \approx q^2 \approx p^2$$

quindi $p \approx \sqrt{n}$ e si cerca nell'intorno della radice di n . Valgono infatti:

$$\left(\frac{p+q}{2}\right)^2 - n = \left(\frac{p-q}{2}\right)^2$$

- Mi dice che $\frac{p+q}{2} > \sqrt{n}$ perché $\left(\frac{p-q}{2}\right)^2 > 0$ sempre, se $p \neq q$.
- Inoltre $\frac{(p+q)^2}{4} - n$ è un quadrato perfetto: $\left(\frac{p-q}{2}\right)^2$.

Posso quindi cercare tra gli interi maggiori di \sqrt{n} fino a trovare z tale che $z^2 - n$ sia un quadrato perfetto. Poniamo

$$z^2 - n = w^2$$

La soluzione non è unica, dunque dobbiamo fare diversi controlli e verificare se i numeri trovati effettivamente fattorizzano n .

Supponiamo dunque di averli trovati, otteniamo:

$$\begin{cases} z = \frac{p+q}{2} \\ w = \frac{p-q}{2} \end{cases} \implies \begin{cases} p = z + w \\ q = z - w \end{cases}$$

La differenza è grande quando cresce come una potenza di n , deve crescere più di un logaritmo.

17.4.4 Attacchi con esponenti (d ed e) bassi

Valori di e e d bassi portano ad accelerare nell'algoritmo, tuttavia se d è piccolo sono possibili attacchi forza bruta.

Problema Se m è piccolo ed anche e lo è, può succedere che $m^e < n$ quindi non ci sia la riduzione in modulo. In tal caso decripto semplicemente calcolando $\sqrt[e]{m}$.

17.4.5 Attacchi a tempo

Ne soffre sia RSA che DH.

Si basano sul tempo di esecuzione dell'algoritmo di decifrazione. Si può quindi determinare d analizzando il tempo impiegato per decifrare.

- Nell'algoritmo delle quadrature successive infatti si esegue una moltiplicazione ad ogni iterazione più una ulteriore moltiplicazione modulare per ciascun bit uguale ad 1 in d .
- Guardando il tempo quindi si può stimare il numero di bit ad 1 in d . Si può risolvere aggiungendo un ritardo casuale alla fine.

17.4.6 Attacchi sulla scelta di e

Abbiamo già detto che un valore e troppo basso comporta il rischio di avere $m^e < n$ e quindi

$$m^e \bmod n = m$$

Generalizzando affermiamo che il problema non si manifesta se

$$e \neq \frac{\phi(n) + k}{k}$$

per ogni valore k che divide $p - 1$ e $q - 1$ (**ENTRAMBI**, attenzione al refuso sul libro di Crittografia), con m ed n coprimi.

Attacco con e troppo piccolo

- Un altro attacco possibile se e è troppo piccolo:
 - supponiamo che ci siano e utenti che hanno scelto lo stesso valore piccolo di e
 - gli e utenti ricevono lo stesso messaggio m

Otteniamo il seguente sistema

$$\begin{cases} c_1 = m^e \bmod n_1 \\ c_2 = m^e \bmod n_2 \\ \dots \\ c_e = m^e \bmod n_e \end{cases}$$

sarà quindi vero che $m < n_i, \forall i \in [1, e]$.

- **Teorema cinese del resto** (da un video di Alessandro Zaccagnini).

Supponiamo di avere un sistema del genere, con n_1 ed n_2 coprimi

$$\begin{cases} x \bmod n_1 = a_1 \\ x \bmod n_2 = a_2 \text{ l'ann} \end{cases}$$

abbiamo un'unica soluzione in modulo $n = n_1 \cdot n_2$ qualunque siano gli interi a_1 e a_2 .

- Ipotiziamo che n_1, n_2, \dots, n_e siano coprimi tra loro (se non lo fossero si potrebbe fattorizzarne qualcuno usando il *MCD*), per il **teorema cinese del resto** sono certo dell'esistenza di un valore m' tale che:

$$\begin{cases} m' < n = n_1 \cdot n_2 \cdot \dots \cdot n_e \\ m' \equiv m^e \bmod n \end{cases}$$

Dato che $m < n_i, \forall i$ so che $m \cdot m \cdot \dots \cdot m = m^e < n_1 \cdot n_2 \cdot \dots \cdot n_e = n$. Quindi la riduzione in modulo non avviene e posso calcolare $m = \sqrt[e]{m'}$.

- Se voglio continuare ad usare e piccolo posso aumentare la dimensione di m introducendo un *padding* casuale, diverso per ogni utente (messaggi non più uguali).

17.4.7 Attacco con lo stesso valore di n (*common modulus*)

Supponiamo che due utenti abbiano generato una chiave pubblica con lo stesso n ma diversi e :

$$\langle e_1, n \rangle, \langle e_2, n \rangle$$

Supponiamo inoltre che $\text{MCD}(e_1, e_2) = 1$ allora

$$\exists r, s \in \mathbb{Z} : e_1 \cdot r + e_2 \cdot s = 1$$

utilizziamo Euclide esteso poichè abbiamo la forma $a \cdot x + b \cdot y = \text{MCD}(a, b)$. Saranno allora $r < 0$ e $s > 0$. Si intercetta $c_i = m^{e_i} \pmod{n}$ e si voglia trovare m :

$$\begin{aligned} m = m^1 &= m^{r \cdot e_1 + s \cdot e_2} = (m^{e_1} \pmod{n})^r \cdot (m^{e_2} \pmod{n})^s = \\ &= (c_1^r \cdot c_2^s) \pmod{n} = ((c_1^{-1})^{-r} \cdot c_2^s) \pmod{n} \end{aligned}$$

Si calcola quindi $c_1^{-1} \pmod{n}$ che esiste se c_1 ed n sono coprimi (se non lo fossero potremmo fattorizzare n). Se non sono coprimi calcola $m = (c_1^{-1})^{-r} \cdot (c_2)^s \pmod{n}$ in tempo polinomiale.

17.5 Cifrari ibridi: uso di RSA ed AES insieme

Il principale scopo della crittografia a chiave pubblica è quello di scambiare le chiavi! Supponiamo che Alice e Bob vogliano parlare tra di loro in maniera sicura...

- **Alice:**

- sceglie una chiave AES $k[\text{session}]$ (detta *chiave di sessione*) e la cifra con la chiave pubblica di Bob
- cifra il messaggio con $k[\text{session}]$ col cifrario AES
- invia i due crittogrammi a Bob (il messaggio cifrato in AES e la chiave cifrata in RSA con la chiave pubblica di Bob).

$$\langle C_{\text{RSA}}(k[\text{session}], k_{\text{BOB}}[\text{pub}]), C_{\text{AES}}(m, k[\text{session}]) \rangle$$

- **Bob:**

- decifra il primo crittogramma con $k_{\text{BOB}}[\text{priv}]$, ottenendo così $k[\text{session}]$
- decifra il secondo crittogramma utilizzando $k[\text{session}]$

NB Al termine della sessione la chiave AES va cancellata. Per ogni nuova sessione di comunicazione va creata una nuova chiave (ad ogni messaggio la cosa è troppo impegnativa).

Problema Un problema di questo approccio è che l'onere di creare la chiave di sessione sta ad Alice. Bob non può essere sicuro che la chiave sia stata generata in modo corretto (creare una chiave rispettando tutti i principi detti richiede risorse)! In genere sarebbe preferibile porre ambo le parti sullo stesso livello. Risolviamo la cosa introducendo il protocollo *Diffie-Hellman*.

Capitolo 18

Protocollo Diffie-Hellman

18.1 Spiegazione

Non è un cifrario, è solo un protocollo per lo scambio della chiave

Questo protocollo permette la creazione di una chiave di sessione nella quale entrambi i lati partecipano in egual misura. Non è un protocollo di scambio di chiavi, la chiave non viaggia mai: si scambiano informazioni che permettono la costruzione della chiave (ogni volta si mettono insieme informazioni private di un interlocutore). Il protocollo ricorre all'algebra modulare e basa la sua sicurezza sul problema del logaritmo discreto.

Passi

- **Primo step.** Alice e Bob si accordano pubblicamente su un numero p primo molto grande (≈ 1000 bit) ed un generatore g di Z_p^* . Ricordiamo che

$$Z_p^* = \{1, 2, \dots, p-1\}$$

che posso esprimere nel seguente modo, grazie al generatore g

$$Z_p^* = \{g^k \bmod p, 1 \leq k \leq p-1\}$$

Ricordarsi che lavorare con un numero primo garantisce l'esistenza di ALMENO un generatore di Z_p^*

- **Individuazione del generatore.** Non esistono algoritmi deterministici per individuare generatori (si usano algoritmi randomizzati). Nel caso in cui Alice e Bob non abbiano la potenza computazionale per ottenere $\langle g, p \rangle$ allora utilizzeranno coppie note, per esempio quelle raccomandate dalle linee guida del NIST: vedremo che ciò non è un problema, nella dimostrazione della correttezza supporremo che la coppia sia nota al crittoanalista.

- **Step successivi.** L'algoritmo prosegue in questo modo:

- Alice sceglie x tale che $1 < x < p - 1$ e calcola

$$A = g^x \bmod p$$

ed invia A a Bob

- Bob sceglie y tale che $1 < y < p - 1$ e calcola

$$B = g^y \bmod p$$

ed invia B ad Alice

- Alice calcola la chiave di sessione:

$$K = B^x \bmod p = \boxed{g^{x \cdot y} \bmod p}$$

- Bob calcola la chiave di sessione:

$$K = A^y \bmod p = \boxed{g^{x \cdot y} \bmod p}$$

Entrambi hanno ottenuto lo stesso numero K .

18.2 Sicurezza del problema del logaritmo discreto

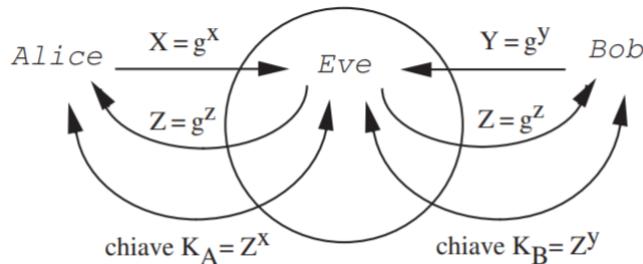
18.2.1 Attacchi passivi

Il crittoanalista conosce il numero primo p , il generatore g , i valori A e B . Per calcolare la chiave di sessione $k[\text{session}]$ bisogna trovare x ed y :

$$\begin{cases} A = g^x \bmod p \\ B = g^y \bmod p \end{cases}$$

siamo di fronte al problema del **logaritmo discreto!** Possiamo solo eseguire un forza bruta ma se p è molto grande allora c'è poco da fare.

18.2.2 Attacchi attivi



Diffie-Hellman è vulnerabile agli attacchi di tipo *man-in-the-middle*. Abbiamo Eve che si finge Alice agli occhi di Bob e Bob agli occhi di Alice:

- Alice invia $A = g^x \bmod p$. Eve intercetta A e lo sostituisce con

$$E = g^z \bmod p$$

Bob riceve E .

- Bob invia $B = g^y \bmod p$. Eve intercetta B e lo sostituisce con

$$E = g^z \bmod p$$

Alice riceve E .

- Eve calcola:

- $K_A = A^z \bmod p = g^{x \cdot z} \bmod p$ per parlare con Alice
- $K_B = B^z \bmod p = g^{y \cdot z} \bmod p$ per parlare con Bob

- Alice calcola $K_A = E^x \bmod p$ per parlare con Bob, ma in realtà parlerà con Eve
- Bob calcola $K_B = E^y \bmod p$ per parlare con Alice, ma in realtà parlerà con Eve

Qualsiasi cosa che Alice mandi a Bob, Eve può decifrarlo, criptare con la chiave di Bob e re-inviarlo. Viceversa per i messaggi di Bob verso Alice. Eve può quindi leggere tutti i messaggi da ambo le parti.

Soluzione Per risolvere questo problema si possono usare le *Certification of Authority*, cioè dei certificati digitali con il quale si firmano i valori A e/o B .

Capitolo 19

Cifrario di El Gamal

19.1 Spiegazione

Il cifrario di El Gamal è un’alternativa al binomio RSA-AES.

Costruzione della chiave pubblica. Bob mette a disposizione la sua chiave pubblica, la costruisce:

- sceglie p molto grande, primo e sceglie g generatore di Z_p^*
- **Scelta della chiave privata.** Bob sceglie x tale che $1 < x < p - 1$. Sarà la chiave privata $k[\text{priv}] = x$, usata da Bob al momento della decifrazione.
- calcola $y = g^x \bmod p$
- ha ottenuto $k[\text{pub}] = \langle p, g, y \rangle$

Cifratura Alice vuole inviare un messaggio m a Bob.

Il messaggio è trattato come un numero tale che: $0 \leq m < p$.

- Si procura la chiave pubblica di Bob $k[\text{pub}] = \langle p, g, y \rangle$
- Sceglie a caso $1 < r < p - 1$ (da tenere segreto)
- Calcola c e d

$$c = g^r \bmod p$$

$$d = m \cdot y^r \bmod p$$

c apparterrà a Z_p^* , ovviamente.

- Allo step precedente Alice ha ottenuto la coppia $\langle c, d \rangle$, che invia a Bob.
 - c contiene ”protetto” l’informazione sul numero casuale r (protetto perchè c’è il problema del logaritmo discreto se volessi calcolare r)

- d è il messaggio m a cui è stata applicata una maschera (messaggio trattato come un numero, moltiplicato per un valore casuale)

Decifrazione Bob deve quindi decifrare il messaggio:

$$m = \frac{d}{c^x} \bmod p = d \cdot \frac{1}{c^x} \bmod p = d \cdot c^{-x} \bmod p$$

x è la chiave privata scelta da Bob! Il calcolo è possibile solo se si conosce la chiave privata.

19.2 Dimostrazione di correttezza

Per dimostrare la correttezza recupero la formula con cui Alice ha calcolato d e c , e la formula con cui Bob ha calcolato y .

$$\frac{d}{c^x} \bmod p = \frac{y^r \cdot m}{c^x} \bmod p = \frac{(g^x)^r \cdot m}{(g^r)^x} \bmod p = m$$

E' sicuro perché Eve conosce $\langle p, g, y, c, d \rangle$, tutto tranne r ed x : conoscere almeno una delle due cose permetterebbe di trovare il messaggio in chiaro m .

Trapdoor

- Se si conoscesse x si potrebbe calcolare

$$m = \frac{d}{c^x} \bmod p$$

- Se si conoscesse r si potrebbe calcolare

$$m = \frac{d}{y^r} = \frac{y^r \cdot m}{y^r} \bmod p$$

NB Importante mantenere r segreto ed usarlo solo una volta!

Capitolo 20

Crittografia su curve ellittiche

Attorno al 1985 Miller (IBM, non lo stesso Miller di Miller-Rabin) e Koblitz (University of Washington) propongono di prendere RSA e DH e cambiare le operazioni in algebra modulare con operazioni su curve ellittiche.

Wait Prima di introdurre il cifrario è necessario entrare nel concetto di *curva ellittica*.

20.1 Vantaggi della crittografia a curve ellittiche

Si sta via via abbandonando la crittografia basata sull'algebra modulare per abbracciare operazioni definite sui punti di una *curva ellittica*:

- le chiavi risultano più piccole a parità di sicurezza;
- fattorizzazione e logaritmo discreto si risolvono in tempi sub-esponenziali.

Gli algoritmi per rompere la cifratura su curve ellittiche rimangono ad ora ancora esponenziali puri (anche qua vale l'approccio delle funzioni *one-way trap-door*).

AES	RSA, DH, El Gamal	ECC
128 bit	3072 bit (n per RSA, p per DH)	256 bit
256 bit	$\approx 15'000$ bit	512 bit

Il fatto è che ECC offre la stessa sicurezza di RSA, ma con un numero di bit decisamente inferiore! Logicamente ci piace di più un cifrario che richiede una minore occupazione di memoria.

20.2 Curve ellittiche

Definizione di campo Il campo K è un insieme non vuoto dotato di operazioni binarie interne, chiamate *addizione* e *sottrazione*, che soddisfano la proprietà associativa, commutativa, di esistenza dell'elemento neutro e di esistenza dell'inverso di ciascun elemento (ad eccezione dello zero per la moltiplicazione).

Forma generale Preso un campo K una curva ellittica è l'insieme dei punti $(x, y) \in K^2$ tale che:

$$y^2 + axy + by = x^3 + cx^2 + dx + e$$

con $a, b, c, d, e \in K$.

Definizione di caratteristica del campo La caratteristica di un campo è il numero di volte per il quale devo sommare l'elemento neutro moltiplicativo 1 per ottenere l'elemento neutro additivo 0.

- Negli interi modulo p la caratteristica è p .
- Nei reali la caratteristica è 0 (non esistono numeri in grado di fare quanto detto).

Forma normale di Weierstrass Se la caratteristica di $K \neq 2, 3$ allora esiste una forma semplificata:

$$y^2 = x^3 + ax + b$$

con $a, b \in K$. Scriviamo quindi:

$$E_K = \{(x, y) \in K^2 \mid y^2 = x^3 + ax + b\} \cup \{0\}$$

Si osservi che introduciamo l'elemento neutro 0 dell'operazione (quella che stiamo costruendo), uniamo all'insieme di punti un elemento 0 che in base al campo può già esistere o meno. Sarà per noi un punto all'infinito.

Gruppo Abeliano All'interno di $E(a, b)$ possiamo definire una operazione interna, dando all'insieme la struttura algebrica di un *gruppo Abeliano*: ci limitiamo a questo, non andiamo a definire un campo vero e proprio (le proprietà presenti sono inferiori rispetto a quelle della definizione di campo).

20.3 Curve ellittiche su $K = \mathbb{R}$

Radice Sia K un campo. Un elemento $a \in K$ si dice radice, o zero, di un polinomio $f(x)$ a coefficienti in K se: $f(a) = 0$ (da <http://progettamatematica.dm.unibo.it/>).

Radice multipla Una radice di un polinomio è detta multipla se ha molteplicità maggiore di 1 (da <http://progettamatematica.dm.unibo.it/>).

Definizione La definizione è quella già introdotta:

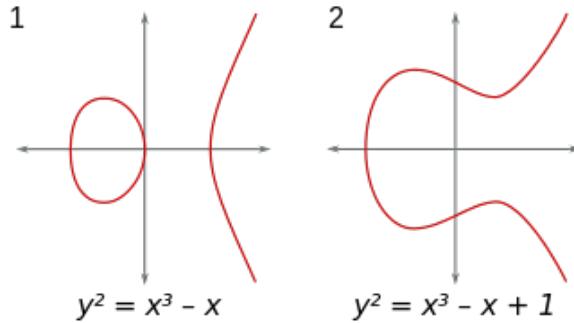
$$E_{\mathbb{R}}(a, b) = \{(x, y) \in \mathbb{R}^2 \mid y^2 = x^3 + ax + b\} \cup \{0\}$$

Per avere un gruppo Abeliano in \mathbb{R} è necessario che $x^3 + ax + b$ non abbia radici multiple, quindi si assume che:

$$4a^3 + 27b^2 \neq 0$$

Questo ci garantisce l'assenza di radici multiple. L'assenza di radici multiple garantisce l'assenza di punti singolari, quindi esiste sempre la tangente esiste in ogni punto della curva ellittica.

Esempi di curve ellittiche



Sulla sinistra una curva a due componenti, sulla destra una curva ad una componente. Si osservi le radici della cubica $x^3 + ax + b$, si hanno tre radici che possono essere:

- 3 reali \implies Curva a due componenti
- 1 reale e 2 complesse coniugate \implies Curva ad una componente

Si noti che le curve ellittiche sono sempre simmetriche rispetto all'asse delle x . Se non imponessimo radici distinte avremmo curve con cuspidi o con nodi, in queste particolari curve la tangente non è detto che esista unica (e questa cosa ci serve).

20.3.1 Punto opposto $-P$

La simmetria viene dal termine y^2 , infatti:

$$P = (x, y) \in E(a, b) \implies y^2 = x^3 + ax + b$$

definiamo dunque:

$$-P = (x, -y) \implies (-y)^2 = y^2 = x^3 + ax + b$$

Dato $P = (x, y) \in E(a, b)$ si dice *opposto* e si indica con $-P$ il punto $-P = (x, -y)$. Per il punto all'infinito si pone $0 = -0$.

20.3.2 Somma sulla curva ellittica

Vogliamo definire un nuovo sistema a chiave pubblica, per fare ciò serve una funzione *one-way trap-door*. Per avere una funzione per prima cosa dobbiamo definire un'operazione sull'insieme dei punti. Definiamo la *somma* (attenzione, non ha nulla a che fare con la somma delle coordinate!)

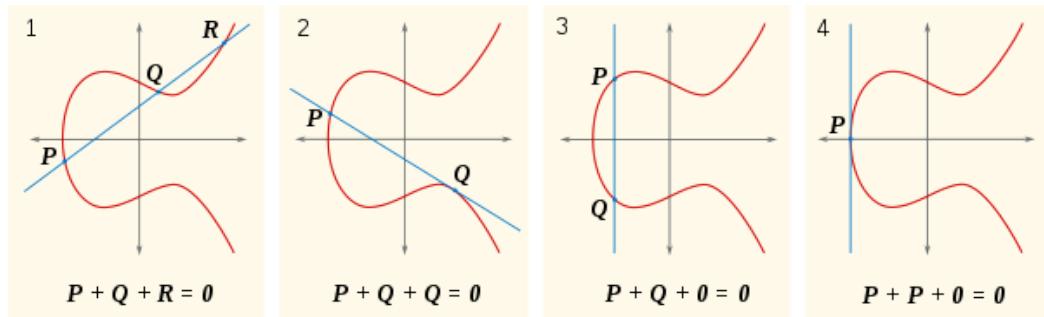
Cosa facciamo? Facciamo l'intersezione tra la curva ellittica e una retta

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

manipolando le equazioni mi trovo con una cubica della x . Le situazioni possibili sono:

- un solo punto reale e due complessi coniugati;
- tre reali.

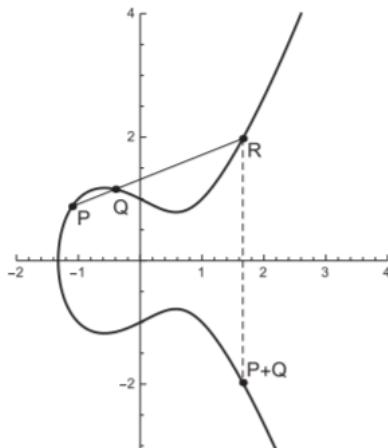
Posso avere al massimo tre punti di intersezione. Posso quindi prendere due punti sulla curva P e Q dato che ho due intersezioni esplicite ce ne sarà una terza per forza, chiamiamolo R . Il risultato di questa somma sarà $-R$ (per definizione).



20.3.2.1 Definizione di somma

Presi $P, Q, R \in E(a, b)$ se P, Q, R sono disposti su una retta si pone:

$$P + Q + R = 0 \implies P + Q = -R$$



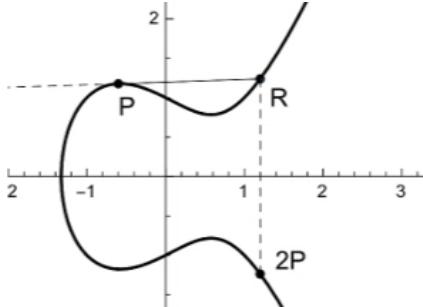
Distinguiamo i vari casi:

- $\mathbf{Q} \neq \pm \mathbf{P}$. I punti $P, Q \in E(a, b)$. Trovo il terzo punto $R \in E(a, b)$ e utilizzo la formula della definizione. Nessun problema poiché $-R \in E(a, b)$
- Se $\mathbf{Q} = -\mathbf{P}$ sommiamo il punto con il suo opposto

$$P + (-P) = -0 = 0$$

Si ricordi che nella nostra operazione l'elemento neutro è il punto all'infinito (si veda la terza figura prima della def.).

- Se $\mathbf{Q} = \mathbf{P}$ si considera la tangente alla curva nel punto P (sempre definita per costruzione in quanto $4a^3 + 2tb^2 \neq 0$) e se ne vede l'intersezione con la curva. Si prende l'opposto del punto di intersezione tra tangente in P e curva (può essere O).



Si noti che anche in questo caso se si fa passare una retta sull'estremo sinistro si ha una retta verticale e quindi la somma può dare il punto all'infinito (si veda la quarta figura prima della def.).

20.3.2.2 Proprietà della somma

- **Chiusura.** $\forall P, Q \in E(a, b) : P + Q \in E(a, b)$

Dalla somma di due elementi otteniamo sempre un terzo elemento appartenente alla curva.

- **Elemento neutro.** $\forall P \in E(a, b) : P + 0 = 0 + P = P$

Per ogni punto appartenente alla curva ellittica esiste l'elemento neutro, ossia un punto che sommato al primo non lo altera.

NB: Prendendo un punto P e tracciando la retta all'infinito passante per P l'altra intersezione è $-P$ quindi il risultato è $-(-P) = P$.

- **Inverso (opposto).**

$$\forall P \in E(a, b) \exists! Q \in E(a, b) : P + Q = 0 = Q + P$$

$$Q = -P \implies P = (x, y), Q = (x, -y)$$

Per ogni punto P appartenente alla curva ellittica esiste uno e un solo punto Q tale che $P + Q = 0$

- **Commutativa.** $\forall P, Q \in E(a, b) P + Q = Q + P$

- **Associativa.** $\forall P, Q, R \in E(a, b) : (P + Q) + R = P + (Q + R)$

20.3.2.3 Formulazione algebrica

Dati

$$P = (x_P, y_P) \quad Q = (x_Q, y_Q) \quad S = (x_S, y_S) = P + Q$$

abbiamo:

- $\mathbf{Q} \neq \pm \mathbf{P}$:

$$\begin{cases} \lambda = \frac{y_Q - y_P}{x_Q - x_P} \\ x_S = \lambda^2 - x_P - x_Q \\ y_S = \lambda(x_P - x_S) - y_P \end{cases}$$

- $\mathbf{P} = \mathbf{Q}$:

$$\begin{cases} \lambda = \frac{3x_P^2 + a}{2y_P} \\ x_S = \lambda^2 - x_P - x_Q \\ y_S = \lambda(x_P - x_S) - y_P \end{cases}$$

NB: se $y_P = 0$ allora $P + P = 0$

- $\mathbf{Q} = -\mathbf{P}$: otteniamo il p.to all'infinito

$$P + Q = 0$$

20.4 Curve ellittiche su campi finiti

In crittografia non possiamo usare le curve su campi reali: la crittografia ha bisogno di un'aritmetica veloce e assolutamente precisa (senza, ad esempio, errori di arrotondamento). Quello che facciamo è prendere curve definite su campi finiti (quindi un numero finito di elementi): vedremo le cosiddette *curve prime*.

- Il campo è l'insieme $\mathbb{Z}_p = \{0, 1, \dots, p-1\}$
- p è numero primo
- La caratteristica di \mathbb{Z}_p è p .
- Imponiamo $p > 3$ per poter utilizzare la forma normale di Weierstrass.

Non abbiamo più una figura continua, ma una nuvola di punti (cit.)! Noi consideriamo solo le curve prime per le quali $4a^3 + 27b^2 \bmod p \neq 0$ (CNS per gruppo Abeliano).

Campo Ridefiniamo quindi la curva:

$$E_p(a, b) = \{(x, y) \in \mathbb{Z}_p^2 \mid y^2 \bmod p = (x^3 + ax + b) \bmod p\} \cup \{0\}$$

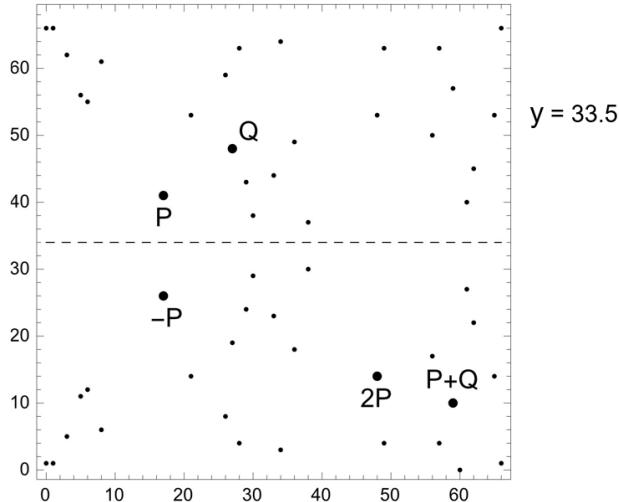
- Si noti che essendo in modulo si lavora sempre nel primo quadrante: $x \geq 0, y \geq 0$.
- La curva sarà ancora simmetrica ma non rispetto a $y = 0$, bensì rispetto $y = \lfloor \frac{p}{2} \rfloor$
- Inoltre tutti i valori saranno in $[0, p-1]$ per il modulo (elementi appartenenti a \mathbb{Z}_p).

- Essendo simmetrica se $P = (x, y) \in E_p(a, b)$ allora

$$-P = (x, p - y) \in E_p(a, b)$$

Se $y^2 \equiv x^3 + ax + b$ allora $(p - y)^2 \equiv p^2 - 2py + y^2 \equiv y^2$ poichè p è lo zero nel campo.

Le formule della somma sono dunque le stesse ma si lavora in modulo (le divisioni diventano moltiplicazioni per l'inverso se non ci sono semplificazioni immediate, l'inverso c'è sempre perché lavoriamo modulo un numero primo).



Esempio Prendiamo $P = (17, 41) \in E_{67}(-1, 1)$. L'elemento è rappresentato in figura. Calcoliamo l'opposto:

$$-P = (17, -41 \bmod 67) = (17, -41 + 67) = (17, 26) \in E_{67}(-1, 1)$$

La condizione per definire un gruppo Abeliano è valida: $(4a^3 + 27b^2) \bmod p \neq 0$

20.4.1 Formulazione algebrica

Siamo nell'insieme \mathbb{Z}_p . Dati

$$P = (x_P, y_P) \quad Q = (x_Q, y_Q) \quad S = (x_S, y_S) = P + Q$$

abbiamo:

- $Q \neq \pm P$:

$$\begin{cases} \lambda = \frac{y_Q - y_P}{x_Q - x_P} \bmod p \\ x_S = (\lambda^2 - x_P - x_Q) \bmod p \\ y_S = (\lambda(x_P - x_S) - y_P) \bmod p \end{cases}$$

- $P = Q$:

$$\begin{cases} \lambda = \frac{3x_P^2 + a}{2y_P} \bmod p \\ x_S = (\lambda^2 - x_P - x_Q) \bmod p \\ y_S = (\lambda(x_P - x_S) - y_P) \bmod p \end{cases}$$

20.4.2 Ordine di una curva ellittica (cardinalità)

Premessa Cosa intendiamo con *residuo quadratico*? Intendiamo un numero intero q se esiste un intero x tale che

$$x^2 \equiv q \pmod{p} \implies x^2 \pmod{p} = q$$

Definizione L'ordine di una curva è il numero di punti che la curva possiede.

$$y^2 = x^2 + ax + b$$

Non esiste una formula precisa per trovare l'ordine: dipende dalla cubica. Prendo un valore $x \in \mathbb{Z}_p$

- Si hanno al massimo p valori per l'ascissa x , quindi mi aspetto circa $2p$ valori per l'ordinata y (si ricordi che nella formula è posto il quadrato). Sommo 1 per considerare il punto all'infinito:

$$2p + 1$$

- Non tutti i valori nel campo hanno una radice, quindi **sono molti meno** di $2p + 1$.
- Il **teorema di Hasse** ci dice che se N è l'ordine della curva prima $E_p(a, b)$ allora:

$$|N - (p + 1)| \leq 2\sqrt{p}$$

Esempio Consideriamo la seguente curva

$$y^2 \equiv (x^3 + 4x + 4) \pmod{5}$$

Calcoliamo i quadrati dei punti del campo, in modo da capire quali sono residui quadratici e quali no.

y	0	1	2	3	4
$y^2 \pmod{5}$	0	1	4	4	1

Osserviamo che i numeri 0, 1, 4 sono residui quadratici. Per la costruzione della curva consideriamo i possibili valori x .

$$\begin{aligned} &\implies y^2 \equiv (x^3 + 4x + 4) \pmod{5} \\ x = 0 &\implies y^2 = 4 \implies (0, 2), (0, 3) \in E_5(4, 4) \\ x = 1 &\implies y^2 = 4 \implies (1, 2), (1, 3) \in E_5(4, 4) \\ x = 2 &\implies y^2 = 0 \implies (2, 0) \in E_5(4, 4) \\ x = 3 &\implies y^2 = 3 \implies \text{Non esistono punti di ascissa 3} \\ x = 4 &\implies y^2 = 4 \implies (4, 2), (4, 3) \in E_5(4, 4) \end{aligned}$$

Abbiamo 7 punti, più il punto all'infinito: l'ordine è 8.

20.4.3 Costruzione di una funzione *one-way trap-door*

C'è una specie di parallelismo tra le operazioni in algebra modulare e le operazioni sulla curva ellittica:

- la **moltiplicazione** (algebra modulare) diventa la **somma tra due punti** (curve ellittiche)
- l'**elevamento a potenza k** (algebra modulare) diventa la **moltiplicazione di un punto per lo scalare k** (curve ellittiche)

$$y^k = y \cdot y \cdots \cdot y \implies kP = P + P + \cdots + P$$

La funzione deve essere one-way, quindi la moltiplicazione deve avere costo polinomiale. Non possiamo fare la somma k volte: non è praticabile, abbiamo una complessità esponenziale rispetto alla dimensione di k (algoritmo pseudopolinomiale, crescita lineare nel valore, crescita esponenziale nella dimensione). Per rendere il numero di operazioni proporzionale al logaritmo del valore ricorriamo a un'adattamento dell'algoritmo delle esponenziazioni veloce: **algoritmo dei raddoppi ripetuti**.

20.4.3.1 Moltiplicazione scalare: algoritmo dei raddoppi ripetuti

Operazione ed esempio Calcolare $Q = kP$ è one-way. Dati i valori k e P è facile calcolare Q (si fa in $O(\log k)$). Consideriamo un esempio:

$$\begin{aligned} Q &= 13P = (1 + 4 + 8)P \\ P &\longrightarrow 2P \longrightarrow 2(2P) = 4P \longrightarrow 2(4P) = 8P \end{aligned}$$

Con 3 raddoppi calcoliamo $8P$. Concludiamo

$$Q = P + 4P + 8P = 13P$$

Formalizzazione

1. Dato $Q = kP$, riscrivo k in notazione binaria:

$$k = \sum_{i=0}^t k_i \cdot 2^i \quad k = (k_t k_{t-1} \dots k_2 k_1 k_0)$$

Mi servono quindi il seguente numero di bit

$$t + 1 = \lfloor \log_2 k \rfloor + 1$$

2. Si calcolano i vari raddoppi

$$2P, 4P, \dots, 2^t P$$

abbiamo t raddoppi ciascuno come raddoppio del punto precedente.

3. Si calcola

$$Q = \sum_{i:k_i=1} (2^i P)$$

in esattamente t somme: $O(t) = O(\log_2 k)$ quindi lineare nella dimensione dell'input.

Operazione inversa Dati P e Q , sulla curva $E_p(a, b)$, trovare il più piccolo k tale che

$$Q = kP$$

Non si conoscono algoritmi polinomiali (neanche sub-exp) per risolvere il problema. Si parla di *logaritmo discreto sulle curve ellittiche*.

20.5 Protocollo Diffie-Hellman su curve ellittiche

Premessa Definiamo l'*ordine di un punto* il più piccolo numero n tale che

$$nB = 0$$

Algoritmo

- Alice e Bob concordano una curva ellittica prima (che soddisfi la condizione di gruppo Abeliano) ed un punto B della curva che abbia ordine molto grande. I punti e le curve sono già consigliate dal NIST, non si devono quindi creare da zero (come già visto nella versione dell'algebra modulare). Sono note e pubbliche.

Il punto B ottenuto è il generatore (ricordarsi cosa facevamo nella versione in algebra modulare: definizione di un intero primo e grande p e di un generatore g).

- L'algoritmo prosegue così:

1. Alice genera un valore $n_A < n$ casuale, calcola

$$P_A = n_A B$$

e lo invia in chiaro a Bob

2. Bob riceve P_A . Genera un valore $n_B < n$ casuale, calcola

$$P_B = n_B B$$

e lo invia in chiaro ad Alice

3. Alice riceve P_B e calcola $S = n_A P_B = n_A n_B B$
4. Bob riceve P_A e calcola $S = n_B P_A = n_B n_A B$

ambo le parti hanno generato lo stesso S e possono quindi generare la stessa chiave di sessione in vari modi (ad esempio $K = x_S \bmod 256$).

Crittoanalista

- Un crittoanalista non può fare nulla perché conosce $E_p(a, b), B, n, P_A, P_B$ ma non può trovare n_A o n_B se non risolvendo il problema del logaritmo discreto su curva ellittica.
- E' sempre possibile un attacco *man-in-the-middle*: segue che la chiave va sempre estratta da un certificato!
- Il sistema col logaritmo discreto su curva ellittica è facile sulla macchina quantistica.

20.6 Scambio di messaggi (equivalente di El-Gamal)

20.6.1 Algoritmo di Koblitz per la trasformazione del messaggio

Per prima cosa bisogna preparare il messaggio in chiaro: deve essere trasformato in punto della curva ellittica.

$$m \rightarrow P_m \in E_p(a, b)$$

Soluzione (sbagliata) Si potrebbe pensare di usare m come x del punto ma potremmo incappare in una y^2 che non è un residuo quadratico, succede $\frac{1}{2}$ delle volte: una tecnica che ha successo solo metà delle volte non va bene!

Soluzione (giusta) Si usa quindi *l'algoritmo di Koblitz* che prende un intero n e restituisce un punto sulla curva. E' un algoritmo randomizzato. Si sceglie h intero tale che $(m + 1) \cdot h < p$, eseguo quindi i tentativi calcolando $x = m \cdot h + i$ con $0 \leq i < h$ e vedo se ottengo un punto valido:

```
// messaggio m, intero h, parametri della curva E
KOBLITZ(m, h, E):
    for(i=0; i < h; i++) {
        x = m*h + i
        z = (x^3 + E.a*x + E.b) mod E.p
        if z è un residuo quadratico {
            y = sqrt(z)
            return (x,y)
        }
    }
    return "failure"
```

- La probabilità di fallimento è $\approx \left(\frac{1}{2}\right)^h$
- La probabilità di successo è circa $\approx 1 - \left(\frac{1}{2}\right)^h$
- Se ci è andata male si modifica il messaggio e si riprova.

Operazione inversa A seguito della decifrazione otteniamo il punto della curva ellittica (x, y) , ma a noi interessa il messaggio m . Quindi:

$$m = \left\lfloor \frac{x}{h} \right\rfloor = \left\lfloor \frac{mh + i}{h} \right\rfloor = \left\lfloor m + \frac{i}{h} \right\rfloor = m$$

20.6.2 Cifratura e decifrazione

Presi una curva $E_p(a, b)$ ed un suo punto B con ordine elevato n , preso inoltre h da usare per Koblitz ogni utente deve generare la sua coppia chiave pubblica-chiave privata.

- **Chiave privata.** Bob estrae $n_{BOB} < n$
- **Chiave pubblica.** Calcola $P_{BOB} = n_{BOB} \cdot B$,

Alice Alice vuole inviare un messaggio cifrato a Bob:

- prende il messaggio m e lo mappa ad un punto della curva $P_m \in E_p(a, b)$ con l'algoritmo di Koblitz
- sceglie un intero a caso r , calcola

$$V = r \cdot B$$

V è un punto che *protegge* r

- calcola $W = P_m + r \cdot P_{BOB}$
- invia la coppia $\langle V, W \rangle$

Attenzione al parallelismo con El-Gamal: in entrambi i casi inviamo a Bob due elementi, di cui uno ha lo scopo di proteggere un valore e l'altro di mascherare il messaggio (che in questo caso rappresentiamo sottoforma di punto della curva)

Bob Bob deve decifrare il messaggio ricevuto da Alice:

- riceve la coppia $\langle V, W \rangle$
- calcola (ricordando che $V = r \cdot B$ e $P_{BOB} = n_{BOB} \cdot B$):

$$\begin{aligned} W - n_{BOB} \cdot V &= P_m + r \cdot P_{BOB} - n_{BOB} \cdot r \cdot B = \\ &= P_m + \cancel{r \cdot n_{BOB} \cdot B} - \cancel{r \cdot n_{BOB} \cdot B} = P_m \end{aligned}$$

- decifra la chiave con la formula detta assieme all'algoritmo di Koblitz.

20.6.3 Sicurezza

La sicurezza è legata al logaritmo discreto nelle curve ellittiche in quanto Eve può:

- partendo da V trovare r e decifrare con:

$$W - r \cdot P_{BOB} = P_m + \cancel{r \cdot P_{BOB}} - \cancel{r \cdot P_{BOB}} = P_m$$

ma per trovare r bisogna risolvere il problema del logaritmo discreto, cioè trovare uno scalare r tale che $V = r \cdot B$

- partendo da P_{BOB} può trovare n_{BOB} e decifrare esattamente come fa Bob, ma anche qui c'è da risolvere $P_{BOB} = n_{BOB} \cdot B$

Si noti che inoltre r è one-time quindi se si dovesse passare a brutare converrebbe comunque andare a ricavare la chiave privata.

- Per attaccare RSA, DH, El Gamal (su algebra modulare) si hanno algoritmi di costo $O(2^{\sqrt{b \log b}})$ con b numero di bit del modulo. Questi attacchi sono basati sull'*index calculus* in quanto si ha una struttura di campo.
- Per attaccare protocolli su curve ellittiche invece si ha un costo $O(2^{\frac{b}{2}})$ con b bit dell'ordine di B , non essendo un campo ma un gruppo Abeliano si pensa che le tecniche usate per l'algebra modulare non verranno mai trasposte anche qui.

Parte VII

Applicazioni moderne della crittografia

Capitolo 21

Identificazione, autenticazione e firma digitale

21.1 Introduzione

Identificazione L’ *identificazione* prevede che un sistema isolato o in rete debba essere in grado di accettare l’identità di un utente che richiede di accedere ai suoi servizi.

Autenticazione L’ *autenticazione* prevede che il destinatario di un messaggio possa essere in grado di accettare:

- l’identità del mittente
- l’integrità del crittogramma ricevuto (che non sia stato modificato, che non sia stato sostituito)

Attenzione

Identificazione \subset Autenticazione

Firma digitale La *firma digitale* ha le seguenti caratteristiche.

- **Non ripudiabilità.** Il mittente non può negare di aver inviato il messaggio m .
- **Autenticazione.** Il destinatario deve essere in grado di autenticare il messaggio.
- Il destinatario non deve poter sostenere che $m' \neq m$ è il messaggio inviato dal mittente.

Tutto questo deve essere verificabile da terzi (per esempio un giudice)! Non sono modalità indipendenti, ciascuna estende le precedenti! Sono funzionalità utilizzate per contrastare gli attacchi attivi. Ci sono realizzazioni sia su cifrari simmetrici che asimmetrici, noi vedremo queste ultime versioni.

21.1.1 Funzioni hash

Una funzione hash: $f : X \rightarrow Y$ è una funzione tale che:

$$n = |X| >> m = |Y|$$

La cardinalità del dominio è molto più ampia della cardinalità del codominio. Essendo non iniettiva il codominio potrà essere partizionato in insiemi di elementi del dominio che condividono lo stesso valore hash (con la stessa *collistione*).

$$\exists X_1, X_2, \dots, X_m \subseteq X$$

Gli insiemi sono disgiunti, tali che:

$$X = X_1 \cup X_2 \cup \dots \cup X_m$$

$$\forall i, x : x \in X_i : f(x) = y$$

- Si vuole che le dimensioni dei vari insiemi X_i siano più omogenee possibile: quindi due elementi estratti a caso da X hanno probabilità circa $\frac{1}{m}$ di avere la stessa immagine y .
- Si vuole che immagini *simili* tra loro appartengano a sottoinsiemi diversi (quindi immagini hash diverse).
- Bisogna gestire le collisioni.

Non ci bastano queste proprietà in crittografia.

21.1.1.1 Funzioni hash *one-way*

Per le applicazioni crittografiche si devono avere le seguenti proprietà ($f : X \rightarrow Y$):

- $\forall x \in X$ è computazionalmente facile calcolare

$$y = f(x)$$

- **Proprietà one-way.**

Per la maggior parte degli $y \in Y$ è computazionalmente difficile determinare

$$x \in X : f(x) = y$$

- **Proprietà claw-free.**

E' computazionalmente difficile determinare $\langle x_1, x_2 \rangle$ in X tale che:

$$f(x_1) = f(x_2)$$

cioè deve essere difficile individuare due elementi che hanno la stessa *collistione*.

21.1.1.2 MD5 (Message Digest, v5)

- Si tratta di una famiglia di algoritmi. Sono stati pubblicati MD2, MD4, MD5. Nei primi due furono trovate falle e Rivest propose MD5 nel 1992.
- Riceve in input un sequenza S di 512 bit e produce un'immagine di 128 bit.
Si *digerisce* la sequenza riducendone la lunghezza ad $\frac{1}{4}$.
- Non resiste alle collisioni e nel 2004 è uscito di scena con debolezze varie. Si considera severamente compromesso (anche dallo stesso Rivest).
Si usa ancora in altri contesti non crittografici.

21.1.1.3 RIPEMD-160

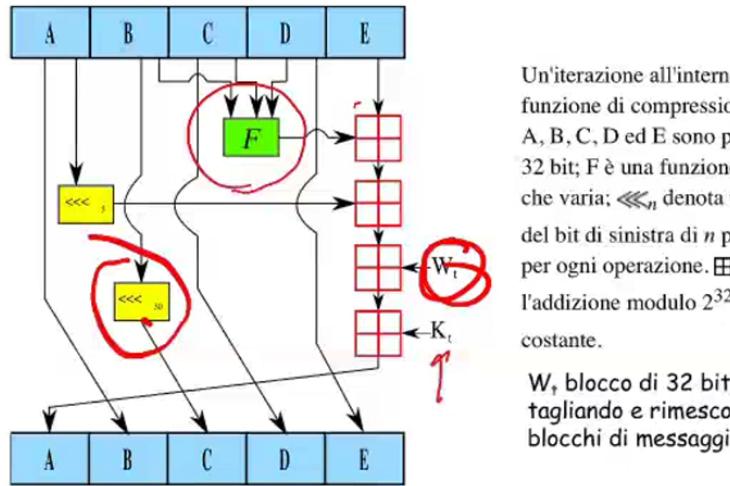
E' la versione matura del *Message Digest* (MD*). Nata nel 1995 produce immagini di 160 bit e non presenta i difetti di MD5.

21.1.1.4 SHA (Secure Hash Algorithm)

- Progettata dal NIST ed NSA nel 1993 (enti statunitensi).
- Opera su sequenze lunghe fino a 2^{64} bit e produce immagini di 160 bit.
- **E' crittograficamente sicura.**
 - La prima versione: SHA-0 conteneva debolezze e fu quindi rivista portando allo SHA-1.
 - Sono poi stati creati SHA-2: altri 4 algoritmi caratterizzati da digest più lunghi.
 - Nel 2007 a causa dei problemi di MD5 e SHA-0 il NIST ha richiesto nuovi standard e nel 2012 è stato selezionato un algoritmo che è diventato SHA-3, pubblicato ufficialmente nel 2015.

SHA-1 a titolo di esempio

- Opera su sequenze fino a $2^{64} - 1$ bit e produce immagini di 160 bit. E' largamente usata nei protocolli crittografici anche se non è più certificata come standard.
- Opera su blocchi di 160 bit contenuti in un buffer di 5 registri da 32 bit ciascuno in cui sono caricati inizialmente dei valori pubblici. Il messaggio m viene poi concatenato con una sequenza di padding che ne rende la lunghezza multipla di 512 bit.
- Il contenuto dei registri varia nel corso dei cicli successivi in cui questi valori si combinano tra loro e con blocchi di 21 bit provenienti da m . Alla fine i registri contengono $\text{SHA-1}(m)$.



Un'iterazione all'interno della funzione di compressione di SHA-1. A, B, C, D ed E sono parole di stato a 32 bit; F è una funzione non lineare che varia; $\ll<_n$ denota una rotazione del bit di sinistra di n posti; n varia per ogni operazione. \oplus denota l'addizione modulo 2^{32} . K_t è una costante.

W_t , blocco di 32 bit ottenuto tagliando e rimescolando i blocchi di messaggio

Il contenuto dei registri varia nel corso dei cicli (all'inizio sono caricati valori fissi e pubblici) in cui questi valori si combinano tra loro e con blocchi di 32 bit provenienti dal messaggio W , nonché con alcuni parametri relativi al ciclo. Alla fine del procedimento (quando è stato letto l'intero messaggio) i registri contengono l'hash SHA1(W)

21.2 Protocollo di identificazione su canali sicuri

Supponiamo di trovarci in un canale sicuro, cioè un canale protetto in lettura e scrittura, inaccessibile ad esterni indesiderati. Supponiamo di avere un utente che vuole accedere alle risorse su un calcolatore: per identificarsi utilizza *username* e *password*.

hash Sono possibili attacchi da utenti interni, che potrebbero accedere al database e leggere le password. Per evitare problemi di questo tipo non si memorizzano mai le password in chiaro, ma un valore *hash*.

Esempio UNIX Quando un utente U fornisce per la prima volta una password P il sistema UNIX associa ad U due sequenze binarie (**LE MEMORIZZA NEL FILE DELLE PASSWORD** al posto della password vera e propria, ricordare quanto visto a Sistemi Operativi):

- S , un seme prodotto mediante generatore pseudocasuale;
- $Q = h(PS)$, hash della concatenazione di P ed S .

Ad ogni successiva connessione il sistema:

- prende l'hash S dal file delle password e la password P' trasmessa dall'utente;
- li concatena ed applica la funzione hash.

Se il risultato della funzione hash è uguale a Q (posto sempre nel file delle password) allora avviene il riconoscimento.



Utilità In questa maniera un accesso al file delle password non rivela informazioni importanti.

21.3 Protocollo di identificazione su canali insicuri

Se il canale è insicuro l'utente che vuole accedere la servizio non può trasmettere la password sul canale stesso: potrebbe essere intercettata durante la sua trasmissione in chiaro! Vediamo quindi un sistema di *identificazione* basato su chiave pubblica e privata.

Sistema proposto Per esempio siano $\langle e, n \rangle$, $\langle d, n \rangle$ le chiavi pubblica e privata di un utente U (stessa struttura vista nell'RSA), che richiede l'accesso ai servizi offerti dal sistema S .

- Il sistema S genera un numero casuale $r < n$ e lo invia in chiaro a U
- L'utente U calcola la *firma di U su r* utilizzando la chiave privata d

$$f = r^d \bmod n$$

invia questo valore al sistema S

- Il sistema S verifica la correttezza del valore ricevuto dall'utente U calcolando:

$$f^e \bmod n = r$$

Verifica l'uguaglianza. In caso di corrispondenza l'identificazione ha successo

Problema: sistemi non fidati Se il sistema S non è *trusted* questo potrebbe inviare un r particolare, anziché randomico, per cercare di carpire informazioni sulla chiave privata! La questione sarà affrontata col protocollo *Zero Knowledge*.

Inversione rispetto all'RSA Le operazioni di criptazione e decifrazione sono invertite (prima viene usata la chiave privata e poi la chiave pubblica) rispetto all'RSA. Ciò è possibile grazie alla seguente proprietà commutativa.

$$(x^e \bmod n)^d \bmod n = (x^d \bmod n)^e \bmod n = x$$

Inoltre f può essere generata solo dall'utente U che possiede la chiave privata $\langle d, n \rangle$.

21.4 Autenticazione su canale insicuro

21.4.1 Protocollo

Per eseguire l'autenticazione invece si usa il MAC: *message authentication code*.

- Mittente e destinatario concordano una chiave segreta k
- Il mittente allega al messaggio un MAC:

$$A(m, k)$$

allo scopo di garantire la provenienza e l'integrità del messaggio. Lo spedisce in coppia col messaggio in chiaro oppure col crittogramma

$$\langle m, A(m, k) \rangle \quad \quad \quad \langle C(m, k'), A(m, k) \rangle$$

- Il destinatario entra in possesso di m , conosce il MAC $A(m, k)$ e ha già concordato la chiave k col suo interlocutore (non è mai passata dal canale insicuro, quindi il crittoanalista non può conoscerla). Ricalcola $A(m, k)$ usando il messaggio m ricevuto e ne confronta il valore col MAC inviato dal mittente.

Se i MAC corrispondono allora l'autenticazione ha successo.

21.4.2 *Message Authentication Code* (MAC)

Il *Message Authentication Code* deve essere una immagine breve del messaggio, generabile solo se si conosce il valore k (lo conoscono solo i due interlocutori). Ci sono varie implementazioni basate su cifrari asimmetrici, simmetrici e funzioni hash one-way. In particolare la versione one-way prevede:

$$A(m, k) = h(mk)$$

con h una *funzione hash one-way*.

Sicurezza E' sicuro in quanto k viaggia nel MAC ed anche se venisse scoperto non è invertibile in maniera semplice. Non può nemmeno sostituire tutto in quanto dovrebbe corredare il messaggio di un nuovo MAC che però non può calcolare essendo all'oscuro del valore di k .

Alternativa Alternativa alle funzioni hash one-way, in un cifrario a blocchi in modalità CBC, è l'uso dell'ultimo blocco come MAC. La cosa è lecitissima in quanto la sequenza ottenuta rappresenta l'intero messaggio.

21.5 Protocolli per la firma digitale

21.5.1 Caratteristiche della firma manuale e di quella digitale

Riflettiamo sulle caratteristiche della firma manuale, da "traslare" nella firma digitale:

- è autentica e non falsabile, prova che chi la ha prodotta è chi ha sottoscritto il documento;
- non è riutilizzabile, è legata al documento su cui è stata apposta;
- il documento non è alterabile, chi ha prodotto la firma è sicuro che questa si riferirà solo al documento sottoscritto nella sua forma originale;
- non può essere ripudiata da chi l'ha apposta costituendo prova legale di un accordo o dichiarazione.

Nella firma digitale si deve tenere conto delle cose già dette (autenticazione in primis) e di un altro dettaglio: essa non è la digitalizzazione della firma manuale, ma un qualcosa a se stante (essa non può essere la digitalizzazione di un documento originale firmato manualmente). Farla dipendere dalla firma manuale significa ridurne il potenziale. Ci sono protocolli che si basano sia su cifrari simmetrici che asimmetrici.

21.5.2 Protocollo 1: messaggio in chiaro e firmato (DH)

L'utente U dispone di una K_{upriv} e K_{upub} . C e D sono funzioni di cifratura e decifratura di un cifrario asimmetrico. La firma funziona:

- **Generazione della firma.**

U genera la firma decifrando il messaggio in chiaro

$$f = D(m, k_{upriv})$$

e spedisce all'utente $\langle U, m, f \rangle$, dove U è l'identificativo dell'utente

- **Verifica della firma.**

L'utente riceve $\langle U, m, f \rangle$, cifra il valore f

$$C(f, K_{upub}) = m$$

e verifica che il risultato sia proprio m

L'indicazione del mittente U consente a V di selezionare la K_{upub} da utilizzare. Per far funzionare questo meccanismo è importante che C e D siano commutative, cioè

$$C(D(m)) = D(C(m)) = m$$

Requisiti della firma digitale Questo protocollo soddisfa i requisiti della firma digitale. Si pensi a quali elementi influenzano la generazione della firma digitale: il messaggio m e la chiave privata (quindi l'utente U , la chiave privata è nota solo a lui).

- E' autentica e non falsabile: K_{upriv} è nota solo al mittente
- Il documento non può essere alterato in quanto m e f sarebbero inconsistenti
- U non può ripudiare la firma perché **può averla prodotta solo lui**
- La firma non è riutilizzabile in quanto f è immagine di m .

21.5.3 Protocollo 2: messaggio cifrato e firmato

Per trasmettere un crittogramma è necessario alterare il protocollo precedentemente introdotto: se cifriamo il crittogramma nella verifica della firma si restituisce il messaggio in chiaro, e la verifica della firma richiede la sola chiave pubblica.

Firma e cifratura Si distingua chiave di U da chiave di V

- U genera la firma sempre a partire dal messaggio m e dalla chiave privata sua:

$$f = D(m, K_{upriv})$$

- Si calcola il crittogramma della firma appena generata con la chiave del destinatario.

$$c = C(f, K_{vpub})$$

- Invia $\langle U, c \rangle$, dove U è l'identificativo dell'utente

Decifrazione e verifica

- L'utente V riceve la coppia $\langle U, c \rangle$ e la decifra con la chiave privata sua:

$$D(c, K_{vpriv}) = f$$

- Verificando la firma (chiave pubblica del mittente) si ottiene anche il messaggio:

$$C(f, K_{upub}) = C(D(m, K_{upriv}), K_{upub}) = m$$

21.5.3.1 Schema applicato all'RSA

Per quanto riguarda le chiavi immaginiamoci il solito schema, sia per U che per V . L'utente U , che vuole scrivere a V , fa i primi passi:

- firma il messaggio $f = m^{d_u} \pmod{n_u}$ (usa la chiave privata di U , cioè $\langle d_u, n_u \rangle$);
- cifra il messaggio $c = f^{e_v} \pmod{n_v}$ (usa la chiave pubblica di V , cioè $\langle e_v, n_v \rangle$);

- spedisce la coppia ottenuta $\langle U, c \rangle$;

L'utente V:

- riceve e decifra $f = c^{d_v} \pmod{n_v}$ (usa la chiave privata di V, cioè $\langle d_v, n_v \rangle$);
- decifra f con k_{upub} , $f^{e_u} \pmod{n_u} = m$ (usa la chiave pubblica di U, cioè $\langle e_u, n_u \rangle$).

Se m è significativo concludo che è autentico.

Osservazioni

- Per la correttezza del procedimento è necessario che $n_u \leq n_v$ affinché sia vero che $f < n_v$ e quindi f possa essere cifrato correttamente. In questo modo tuttavia V non può rispondere in quanto dovrebbero avere $n_u = n_v$ ma è poco sicuro e probabile.
- Possiamo ovviare quindi dando ad ogni utente due chiavi distinte:
 - una per la firma $\langle H$
 - una per la cifratura $\rangle H$

con H pubblico e molto grande. NB: si può attaccare se ci si procura la firma di un utente su un messaggio apparentemente privo di senso

21.5.3.2 Attacco con giochi algebrici

Supponiamo che un utente U abbia l'abitudine di inviare una risposta automatica (ack) al mittente ogni volta che riceve un messaggio m , e che questo messaggio sia il crittogramma della firma di U su m .

Crittoanalista Un crittoanalista attivo X può decifrare i messaggi inviati a U:

- X intercetta c firmato e crittografato inviato da V a U , lo rimuove dal canale e lo rispedisce a U con identificativo X . (cancella $\langle V, c \rangle$ ed invia $\langle X, c \rangle$)
- U spedisce un ack ad X in risposta
- X usa l'ack per risalire al messaggio m applicando le funzioni del cifrario con le chiavi pubbliche di V e di U .

In particolare succede:

- V invia c a U :

$$\begin{cases} c = C(f, K_{upub}) \\ f = D(m, K_{vpriv}) \end{cases} \implies \langle V, c \rangle$$

- X intercetta la coppia, la rimuove dal canale prima che possa essere consegnata al destinatario, invia all'utente U la coppia $\langle X, c \rangle$

- U riceve la coppia $\langle X, c \rangle$ decifra c :

$$f = D(c, K_{upriv})$$

e verifica la firma con la chiave pubblica di X ottenendo:

$$m' = C(f, K_{xpub})$$

Il messaggio m' ottenuto è privo di senso ($m' \neq m$) perchè la generazione della chiave non è avvenuta con la chiave di X , ma con quella di V .

- Nonostante l'inconsistenza del messaggio l'utente U l'ack c' ad X :

$$f' = D(m', K_{upriv}) \implies c' = C(f', K_{xpub})$$

- **Giochini algebrici.**

Il crittoanalista X usa c' ricevuto da U per risalire ad m : decifra (con la chiave privata del crittoanalista)

$$c' : D(c', K_{xpriv}) = f'$$

e verifica (con la chiave pubblica del destinatario)

$$f' : C(f', K_{upub}) = m'$$

Da m' ricava (chiave privata del crittoanalista)

$$f : D(m', K_{xpriv}) = f$$

verifica (chiave pubblica del vero mittente)

$$f : C(f, K_{vpub}) = m$$

Attraverso questi giochi l'utente ha ottenuto f , e a quel punto tutto è in discesa: basta conoscere la chiave pubblica di V ! Per evitare l'attacco si deve evitare l'ack automatico.

21.5.4 Protocollo resistente agli attacchi

Si evita di firmare il messaggio! Si crea un hash del messaggio (SHA) con una funzione hash one-way e vi si appone la firma sopra. La firma non è quindi più soggetta ai giochi algebrici appena visti.

Generazione della firma e cifratura

- il mittente U calcola $h(m)$ e genera: $f = D(h(m), K_{upriv})$
- calcola separatamente: $c = C(m, K_{vpub})$
- spedisce a V la tripla: $\langle U, c, f \rangle$

Decifrazione e verifica

- V riceve e verifica $\langle U, c, f \rangle$
- decifra il crittogramma $c : m = D(c, K_{upriv})$
- calcola $h(m)$ (conosce la funzione *hash* come il suo interlocutore) e $C(f, K_{upub})$ (conosce la chiave pubblica dell'interlocutore) e ne controlla l'uguaglianza, se sono uguali il messaggio è autentico

La firma si calcola più velocemente.

21.5.5 Attacchi man-in-the-middle

Le chiavi di cifratura sono pubbliche e non richiedono un incontro diretto per il loro scambio. Un crittoanalista attivo può quindi intromettersi nella fase iniziale comportandosi come V agli occhi di U e come U agli occhi di V .

- U richiede a V la sua chiave pubblica
- X intercetta la risposta con K_{vpub} e la sostituisce con la sua chiave pubblica K_{xpub}
- X si pone in attesa dei crittogrammi spediti da U a V cifrati mediante K_{xpub}
- X rimuove dal canale ciascuno dei crittogrammi, li decripta e li cifra con K_{vpub} trafugata all'inizio, rispedendola a V
- U e V non si accorgono di nulla se il tutto è fatto velocemente

La cosa si risolve introducendo la *Certification Authority*.

21.6 Certification Authority - CA

Sono infrastrutture/enti che garantiscono la validità delle chiavi pubbliche e ne regolano l'uso gestendo la distribuzione delle chiavi a chi vuole comunicare. La CA autentica l'associazione emettendo un certificato digitale

\langle utente, chiave pubblica \rangle

Il certificato consiste della chiave pubblica e di una lista di informazioni relative al suo proprietario opportunamente firmate dalla CA. Mantiene quindi un archivio accessibile a tutti ma protetto da scritture non autorizzate. La chiave della CA è nota agli utenti che la mantengono protetta e la utilizzano per verificare la firma della stessa CA.

21.6.1 Contenuto del certificato digitale

Si compone principalmente di:

- indicazione del formato (numero di versione)
- nome della CA che lo ha rilasciato
- numero seriale che lo individua univocamente nella CA emittente

- specifica dell'algoritmo usato dalla CA per creare la firma digitale
- periodo di validità
- nome ed altre informazioni dell'utente a cui si riferisce il certificato
- nome dell'algoritmo, parametri e chiave pubblica usati dall'utente per cifr. e firm.
- firma della CA su tutto quanto

Quindi:

- Se U vuole comunicare con V può richiedere K_{vpub} alla CA oppure direttamente a V e poi lo convalida tramite la CA.
- Dato che U conosce K_{CA-pub} può controllarne l'autenticità e la validità.
- Ci si può mettere nel mezzo solo falsificando la certificazione ma si assume che la CA sia fidata.
- Esistono varie CA organizzate ad albero, la verifica quindi è più complicata in quanto si cerca il primo antenato comune tra le CA di U e di V . Ogni utente poi mantiene in cache una copia dei certificati richiesti ed una copia di K_{CA-pub} per non doverla richiedere.

21.7 Protocollo finale con certificazione digitale

Mittente Il mittente U :

1. si procura $cert$ di V (dove è presente la chiave con cui calcoliamo c)
2. calcola $h(m)$ (con la sua chiave privata) e firma:

$$f = D(h(m), K_{upriv})$$

3. calcola c (usando la chiave pubblica del destinatario, ottenuta dal certif. digitale):

$$c = C(m, K_{vpub})$$

4. spedisce la terna $\langle cert_U, c, f \rangle$ ($cert_U$ contiene K_{upub} , quindi non è necessario trasmetterla separatamente)

Destinatario Il destinatario V :

- riceve $\langle cert_U, c, f \rangle$ e verifica l'autenticità di $cert_U$ (e di K_{upub}) tramite la CA.
- decifra il crittogramma:

$$m = D(c, K_{priv})$$

- verifica l'autenticità della firma:

$$C(f, K_{upub}) = h(m)$$

Attenzione La catena ha un punto debole: i certificati non più validi; è quindi cruciale controllare periodicamente con la CA i certificati scaduti.

Capitolo 22

Protocollo conoscenza zero (*Zero Knowledge*)

22.1 Idea Generale

Un protocollo *zero knowledge* è un protocollo che permette ad un utente detto *Prover* (il dimostratore) di dimostrare di avere una certa conoscenza ad un *Verifier* (il verificatore) senza inviare alcuna informazione se non l'evidenza di avere questa conoscenza.

Esempio Facciamo un esempio concreto: supponiamo che Peggy dica di poter sapere se il numero di granelli di sabbia di cui si compone una spiaggia è pari o dispari, e che Victor voglia controllare la veridicità di questa affermazione senza arrivare alla totale certezza ma avvicinandosi tramite una probabilità prossima ad 1. Supponiamo che Peggy risponda:

- 1 se il numero è dispari
- 0 se il numero è pari

Immaginiamo il seguente algoritmo:

```
for i = 1 to k { // k numero di sfide che Peggy deve superare
    <P si volta>
    <V sceglie e in {0, 1} casualmente> // un po' come lanciare una moneta
    if (e == 0)
        <V rimuove un granello di sabbia>
    <V chiede a P il nuovo b[i]>
    if ((e == 0 && b[i] != b[i-1]) || (e == 1 && b[i] == b[i-1]))
        <vado alla prossima iterazione>
    else
        <P è un imbroglio -> STOP>
}
```

Facciamo k test per assicurarci che Peggy stia dicendo il vero. Se una sfida viene perduta allora Peggy è un imbroglio, e ci fermiamo: deve essere in grado di vincere tutte le sfide!

Probabilità La probabilità, al passo k, che Peggy sia un impostore è pari a:

$$P(\text{Peggy è un impostore}) = \frac{1}{2^k}$$

altresì la probabilità che Peggy dica il vero è:

$$P(\text{Peggy dice il vero}) = 1 - \frac{1}{2^k}$$

Se il Prover è onesto non ha problemi a rispondere a tutte le iterazioni in maniera corretta, se non lo è può solo provare ad indovinare, quindi la probabilità di vincere imbrogliando è pari a quella di indovinare una sequenza di k bit casuali. Quanto detto, attenzione, non è una dimostrazione rigorosa.

22.1.1 Proprietà generali

Quali sono le proprietà che deve avere un protocollo per poter essere definito *protocollo a conoscenza zero*?

- **Completezza.**

Se ciò che dice P è vero e V è onesto allora V deve sempre accettare la dimostrazione.

- **Correttezza.**

Se P dice il falso allora V può essere ingannato con probabilità $\leq \frac{1}{2^k}$, con k scelto da V.

- **Conoscenza zero.**

Se l'affermazione di P è vera V (anche se disonesto) non può acquisire alcuna informazione se non il fatto che P ha la conoscenza che dice di avere.

22.2 Protocollo di identificazione Fiat-Shamir

E' un protocollo di identificazione nel quale P dimostra a V la sua identità senza svelare altre informazioni. Si basa sulla difficoltà di invertire la funzione potenza nell'algebra modulare, ovvero di calcolare la radice in modulo.

Precisamente Abbiamo la formula

$$t = s^2 \bmod n$$

Dati s, n (n composto) è facile trovare t (calcolare la potenza). Dati t ed n è difficile trovare s , cioè calcolare la radice in modulo.

$$s = \sqrt{t} \bmod n$$

Il Verifier conosce $< t, n >$, il Prover deve convincere il Verifier di conoscere s . A situazione normale il Verifier riceve s dal prover e calcola t in tempo polinomiale: se il t calcolato coincide col t che già conosceva allora è sicuro che il Prover conosce s . Nel caso dello Zero Knowledge vogliamo dare al Prover una "quasi certezza" di questo senza che s venga trasmesso sul canale.

22.2.1 Generazione della chiave da parte del Prover

Il prover genera la chiave, sia quella pubblica che quella privata.

- P genera p e q numeri primi (grandi)
- P calcola $n = p \cdot q$ (otteniamo un numero composto)
- P sceglie s numero casuale tale che $s < n$ (s sarà il suo segreto)
- P calcola $t = s^2 \pmod{n}$

La chiave pubblica sono i valori conosciuti dal Verifier: $\langle t, n \rangle$. La chiave privata è ciò che è conosciuto esclusivamente dal Prover: $\langle p, q, s \rangle$. Nel protocollo il Verifier vuole assicurarsi che il Prover conosca la chiave privata senza accedere alla chiave privata.

22.2.2 Autenticazione

L'algoritmo di autenticazione prevede k iterazioni con k scelto da V. Dopo k passi il V si convince che P conosce la radice (con la probabilità che abbiamo detto). Arriverà a questa conclusione senza entrare in contatto con i valori segreti.

1. P genera un intero $r < n$, calcola

$$u = r^2 \pmod{n}$$

e comunica u a V

2. V sceglie casualmente $e \in \{1, 0\}$ e lo comunica a P
3. P calcola $z = r \cdot s^e \pmod{n}$ e lo comunica a V
 - $e = 0 \implies z = r \pmod{n} = r$ (poichè $r < n$)
 - $e = 1 \implies z = r \cdot s \pmod{n}$
4. V calcola $x = z^2 \pmod{n}$ e controlla se $x == (u \cdot t^e \pmod{n})$
 - se sono uguali si passa alla prossima iterazione
 - se sono diversi P è un impostore

Si noti che se tutto è corretto:

$$x = z^2 = (r \cdot s^e)^2 \pmod{n} = r^2 \cdot s^{2e} \pmod{n} = u \cdot t^e \pmod{n}$$

Se facciamo calcoli distinti otteniamo:

$$\begin{aligned} x &= z^2 \pmod{n} = r^2 \pmod{n} = u & e &= 0 \\ x &= z^2 \pmod{n} = (r \cdot s)^2 \pmod{n} = r^2 \cdot s^2 \pmod{n} = u \cdot t \pmod{n} & e &= 1 \end{aligned}$$

22.2.3 Dimostrazione di completezza e correttezza

Completezza La completezza si ottiene con le ultime due formule poste nella pagina precedente! Se P è onesto allora passa tutte le prove e V accetta la dimostrazione.

Correttezza Supponiamo che P sia disonesto: non conosce s e l'unica cosa che può fare è tentare di indovinare i valori di e decisi da V .

- **Se prevede $e = 1$.**

Sceglie r casualmente ma invia $u = \frac{r^2}{t} \pmod n$, successivamente manderà $z = r \pmod n$. V andrà quindi a verificare: $x = z^2 = r^2$,

$$u \cdot t = \frac{r^2}{t} \cdot t = r^2 \implies x = u \cdot t \pmod n$$

- **Se prevede $e = 0$.**

Esegue l'algoritmo corretto infatti $x = z^2 = r^2$,

$$u \cdot t^e = u = r^2 \implies x = u \pmod n$$

se prevede bene le formule ottenute sono quelle già calcolate. La probabilità con cui P è in grado di prevedere il corretto valore di e equivale a $\frac{1}{2^k}$, dove k è il numero di iterazioni/challenge a cui viene sottoposto P da V .

22.2.4 Dimostrazione della proprietà Zero Knowledge

Non abbiamo fatto la dimostrazione

22.2.5 Sicurezza a confronto con altri protocolli

Questo protocollo di autenticazione è un protocollo sicuro da entrambi i lati, sia per chi si deve far riconoscere che per chi autentica le richieste. Pensiamo ad una generica autenticazione tramite crittografia asimmetrica:

- il client si vuole autenticare presso un server
- il server dispone della chiave pubblica del client, chiede quindi al client di firmare un messaggio r casuale
- il client riceve r , lo firma e lo restituisce al server
- il server verifica la firma e se è corretta l'autenticazione è avvenuta

In questo scenario ci si deve fidare del server in quanto si prende un messaggio arbitrario e lo si firma con la propria chiave privata, questo tuttavia potrebbe essere problematico in quanto possono esistere degli input malevoli che permettono di ottenere informazioni circa la chiave privata del client. Un protocollo a zero knowledge invece pone entrambi i lati della comunicazione sullo stesso piano di diffidenza.

Capitolo 23

Protocollo SSL (*Secure Socket Layer*)

23.1 Introduzione

L'utente U desidera accedere via Internet a un servizio offerto dal sistema S .

SSL (*Secure Socket Layer*) è alla base dei protocolli più diffusi nelle comunicazioni sicure. Lo portiamo come esempio perchè combina molte cose viste durante il corso.

- Oggi non si utilizza la versione che introdurremo, ma il TLS (*Transport Layer Security*, variante con qualche modifica).
- Garantisce **confidenzialità** e **affidabilità** delle comunicazioni su internet proteggendole da intrusioni, modifiche e falsificazioni.
- Sviluppato inizialmente da *Netscape* per mettere in sicurezza HTTP, è progettato in modo da permettere la comunicazione tra computer che non conoscono le reciproche funzionalità.
- Garantisce la confidenzialità tramite l'uso di un sistema ibrido:
 - cifrario asimmetrico per la costruzione e lo scambio delle chiavi di sessione;
 - cifrario simmetrico per la cifratura delle comunicazioni.
- Garantisce l'autenticazione dei messaggi accertando l'identità dei due partner attraverso sia un cifrario asimmetrico che tramite certificati digitali con un MAC (*Message Authentication Code*, che usa una funzione hash one way crittograficamente sicuro).
- Si trova tra il TCP (trasporto) e HTTP (applicazione) ma è totalmente indipendente dal protocollo applicativo sovrastante (HTTPS = HTTP su SSL).

23.2 Livelli in cui SSL è organizzato

SSL è organizzato in:

- **SSL record.**

Livello più basso, connesso direttamente al protocollo di trasporto. Ha l'obiettivo di incapsulare i dati spediti dai livelli superiori assicurando confidenzialità ed integrità.

- **SSL handshake.**

Instaura e mantiene i parametri poi usati da SSL record. Si permette a utente e sistema di:

- autenticarsi;
- negoziare gli algoritmi di cifratura e firma;
- stabilire le chiavi per i signoli algoritmi crittografici e per il MAC.

Il tutto permette di creare un canale di comunicazione sicuro, affidabile e autenticato tra utente e sistema, all'interno del quale SSL Record fa viaggiare i messaggi. Abbiamo uno scambio di messaggi preliminare (*handshake*) per la creazione del canale sicuro: attraverso questi messaggi il sistema S e l'utente U

- si identificano a vicenda, e
- cooperano nella costruzione delle chiavi segrete da impiegare nelle comunicazioni successive.

Vediamo singolarmente le fasi.

23.3 Fasi in SSL handshake

23.3.1 Messaggio *client hello* inviato dall'utente U

L'utente U invia al sistema S un messaggio con il quale:

- si richiede la creazione di una connessione SSL;
- specifica i cifrari e meccanismi che supporta e le prestazioni di sicurezza richieste;
- invia una sequenza di byte casuali.

Tutto questo è mandato in chiaro!

Esempio Un esempio di *client hello* è la *cypher suite*.

SSL_RSA_WITH_AES_CBC_SHA1

Vuole utilizzare i seguenti cifrari:

- **RSA** per lo scambio delle chiavi di sessione;

- **AES** come cifratura simmetrica;
- **CBC** come composizione dei blocchi;
- **SHA1** come funzione hash per il MAC.

Questo è un esempio di *cipher suite*. Il sistema risponderà nella fase successiva: *server hello*.

23.3.2 Messaggio *server hello* inviato dal server *S*

Il sistema *S* riceve il messaggio dell'utente *U*.

- Seleziona una cipher suite che anche lui supporta (cerca di soddisfare le richieste dell'utente *U*)
- Invia un messaggio all'utente *U* dove specifica la sua scelta.
- Appende dei byte casuali alla risposta.

Anche questo è mandato in chiaro!

NB Se *U* non riceve il *server hello* allora interromperà la comunicazione.

23.3.3 Autenticazione (creazione certificato da parte del sistema *S*)

Il sistema *S* si autentica con *U* inviandogli il proprio certificato digitale (e gli eventuali altri certificati fino al primo nodo comune nella catena delle CA).

NB Se i servizi offerti da *S* devono essere protetti negli accessi anche *S* può richiedere a *U* di autenticarsi inviando il suo certificato digitale. Avviene raramente in quanto la maggior parte degli utenti non ha un proprio certificato e in genere ci si accerta dell'identità di un utente in un secondo modo (autenticazione sul sito web ad esempio).

23.3.4 Messaggio *server hello done*

Messaggio con il quale il server *S* sancisce la fine degli accordi sulla cipher suite ed i parametri crittografici associati.

23.3.5 Controllo del certificato da parte dell'utente *U*

L'utente *U* accerta l'autenticità del certificato ricevuto dal sistema *S* tramite la data, tramite la CA che lo ha firmato, ecc. Estraе la chiave pubblica dal certificato. L'utente *U*:

- costruisce il *pre-master secret* costituito da una nuova sequenza casuale di byte;
- lo cifra con la chiave pubblica estratta dal certificato.
- spedisce il crittogramma al sistema *S*.

23.3.6 Costruzione del *master secret* presso l'utente *U*

Sempre l'utente *U* costruisce il *master secret* partendo da:

- il *pre-master secret* (cifrato con RSA, si usa la chiave pubblica presente nel certificato di *S*);
- i byte casuali di *client hello* e *server hello* (*client hello* lo ha di suo, mentre *server hello* è stato inviato dal server).

Applica a tutte queste sequenze delle funzioni hash one-way secondo una combinazione opportuna. Il nuovo valore ottenuto è il *master secret*.

23.3.7 Ricostruzione del *master secret* presso il server *S*

Anche il sistema *S* si calcola localmente il master secret. Può farlo perchè possiede:

- il *pre-master secret*, appena ricevuto dall'utente *U* e decriptato con la sua chiave privata
- i byte casuali di *client hello* e *server hello* (*client hello* lo ha ricevuto dall'utente *U*, mentre *server hello* lo ha di suo)

Entrambi gli utenti hanno il *master secret*!

23.3.7.1 Extra: Autenticazione dell'utente *U* (se richiesto certificato a *U*)

- Se all'utente *U* è richiesto un certificato ed egli non lo possiede il sistema *S* interrompe l'esecuzione del protocollo.
- Altrimenti *U* invia il proprio certificato allegando *master secret* e la SSL-history (tutti i messaggi scambiati fino a quel momento) firmate con la sua chiave privata.
- Il sistema *S* controlla certificato ed SSL-history: **nel caso in cui emergano ambiguità la comunicazione viene interrotta.**

23.3.8 Messaggio *finished* inviato sia dall'utente che dal server

E' il primo messaggio protetto da *master secret* e *cipher suite* accordati. Il messaggio è costruito dall'utente *U* e inviato al sistema *S*, poi costruito dal sistema *S* ed inviato all'utente *U*. Il messaggio ha la stessa struttura ma cambiano i dati (la history).

Costruzione del messaggio La costruzione avviene:

- concatenazione di *master secret*, messaggi di handshake scambiati fino al momento attuale, e l'identità del mittente (*U* o *S*);
- la stringa ottenuta viene trasformata applicando funzioni hash, sia con SHA-1 che con MD5.

I due messaggi (quello trasmesso dall'utente U e quello successivamente trasmesso dal server S) sono diversi in quanto il sistema S aggiunge anche la *finished* ricevuta da U (che adesso fa parte della history). Non sono ovviamente invertibili perché costruiti tramite hash.

Uso del master secret Il master secret viene usato per creare le chiavi:

- chiave segreta per il cifrario simmetrico
- chiave per l'autenticazione del messaggio (MAC)
- *initialization vector* per il CBC

Le terne di chiavi di U ed S sono diverse tra loro ma note ad entrambi, ciascuno usa la propria, quindi aumenta la sicurezza.

23.4 SSL Record

Successivamente i parametri vengono passati al SSL Record che si occupa del vero e proprio dialogo dei dati suddividendo i dati in blocchi, aggiungendo un MAC ad ogni blocco, cifrato con il protocollo simmetrico scelto e trasmesso al protocollo di trasporto sottostante. Il destinatario esegue il contrario.

23.5 Sicurezza di SSL

- Nei passi di *hello* i due partner creano e si inviano sequenze casuali per costruire la master secret, questo risulta in chiavi diverse ogni volta. Un crittoanalista non può riutilizzare i messaggi catturati sul canale per sostituirli a S successivamente.
- L'SSL record successivamente numera ogni blocco in maniera incrementale e lo autentica tramite MAC. Un blocco non può essere modificato in quanto il MAC è hash di:
 - contenuto
 - numero sequenziale del blocco
 - chiave del MAC
 - altre stringhe note e fissate a priori
- MAC e messaggio sono cifrati quindi prima di poterli alterare va forzato il cifrario.
- Non si possono eseguire man-in-the-middle in quanto si usano i certificati.
- Il pre-master secret è inviato cifrato con chiave pubblica quindi è sicuro.
- Quindi solo S , proprietario della chiave privata associata al certificato può recuperare la pre-master key e quindi entrare in comunicazione con U .

- Anche U può essere autenticato se necessario. Se è richiesto ma non avviene la comunicazione salta. L'opzionalità di questa autenticazione ha reso usato questo protocollo in quanto può essere autenticato in altri modi, magari lato applicazione.
- Le sequenze sono generate a partire da byte casuali quindi una sua predicitività metterebbe a rischio tutto.
- Il messaggio finished contiene tutta la history e quindi permette un ulteriore controllo finale. È sicuro almeno quanto il cifrario più debole della cipher suite, ma non ci sono cifrari deboli lasciati dentro.

Client

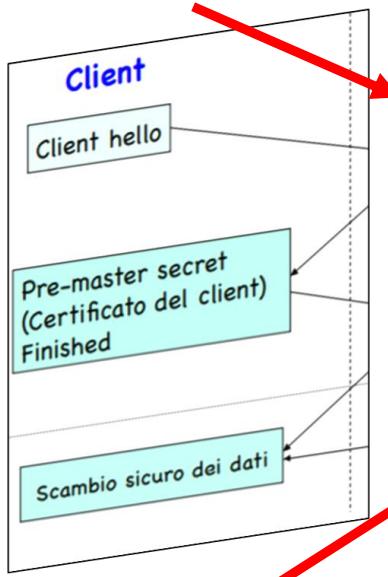
Server

Messaggio *client hello*

L'utente U invia al sistema S un messaggio con il quale:

- si richiede la creazione di una connessione SSL;
- specifica i cifrari e meccanismi che supporta e le prestazioni di sicurezza richieste;
- invia una sequenza di byte casuali.

Tutto questo è mandato in chiaro!



Messaggio *server hello*

Il sistema S riceve il messaggio dell'utente U .

- Seleziona una cipher suite che anche lui supporta (cerca di soddisfare le richieste dell'utente U)
- Invia un messaggio all'utente U dove specifica la sua scelta.
- Appende dei byte casuali alla risposta.

Anche questo è mandato in chiaro!

Autenticazione

Il sistema S si autentica con U inviandogli il proprio certificato digitale (e gli eventuali altri certificati fino al primo nodo comune nella catena delle CA).

NB Se i servizi offerti da S devono essere protetti negli accessi anche S può richiedere a U di autenticarsi inviando il suo certificato digitale. Avviene raramente in quanto la maggior parte degli utenti non ha un proprio certificato e in genere ci si accerta dell'identità di un utente in un secondo modo (autenticazione sul sito web ad esempio).

Messaggio *server hello done*

Messaggio con il quale il server S sancisce la fine degli accordi sulla cipher suite ed i parametri crittografici associati.

Controllo da parte del client

L'utente U accetta l'autenticità del certificato ricevuto dal sistema S tramite la data, tramite la CA che lo ha firmato, ecc. Estraе la chiave pubblica dal certificato. L'utente U :

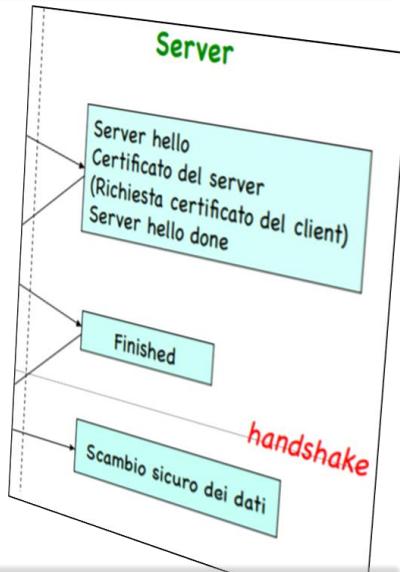
- costruisce il *pre-master secret* costituito da una nuova sequenza casuale di byte;
- lo cifra con la chiave pubblica estratta dal certificato.
- spedisce il crittogramma al sistema S .

Costruzione del *master secret*

Sempre l'utente U costruisce il *master secret* partendo da:

- il *pre-master secret* (cifrato con RSA, si usa la chiave pubblica presente nel certificato di S);
- i byte casuali di *client hello* e *server hello* (*client hello* lo ha di suo, mentre *server hello* è stato inviato dal server).

Applica a tutte queste sequenze delle funzioni hash one-way secondo una combinazione opportuna. Il nuovo valore ottenuto è il *master secret*.



Ricostruzione del *master secret*

Anche il sistema S si calcola localmente il master secret. Può farlo perché possiede:

- il *pre-master secret*, appena ricevuto dall'utente U e decriptato con la sua chiave privata
- i byte casuali di *client hello* e *server hello* (*client hello* lo ha ricevuto dall'utente U , mentre *server hello* lo ha di suo)

Entrambi gli utenti hanno il *master secret*!

Messaggio *finished*

E' il primo messaggio protetto da *master secret* e *cipher suite* accordati. Il messaggio è costruito dall'utente U e inviato al sistema S , poi costruito dal sistema S ed inviato all'utente U . Il messaggio ha la stessa struttura ma cambiano i dati (la history).

Capitolo 24

Quantum Distribution Key (QDK)

24.1 Introduzione

- Il protocollo QDK è un protocollo alternativo a Diffie-Hellman per lo scambio delle chiavi.
- Permette di scambiare lunghe chiavi da utilizzarsi con one-time pad.
- Inoltre è già utilizzato in quanto non necessita dell'utilizzo di computer quantistici e se in futuro la tecnologia quantistica dovesse evolversi comunque questo protocollo sarebbe quantistico-resistente oltre che resistente al collasso della classe NP.

Questo protocollo inoltre permette di accertare una eventuale intrusione, eventualmente si butta via la chiave, proprio per questo motivo non si può usare per scambiare messaggi.

NB Esiste poi la crittografia post-quantistica che si occupa di cercare sistemi a chiave pubblica inattaccabili anche da macchine quantistiche. Ci si basa su problemi NP-completi che si sanno essere difficili anche con una eventuale supremazia quantistica.

24.1.1 Principi della meccanica quantistica

Sfruttiamo i seguenti principi della meccanica quantistica.

- **Sovrapposizione degli stati**

Un sistema quantistico si può trovare in più di uno stato allo stesso tempo. Matematicamente si intende una combinazione lineare dei possibili stati (i coefficienti sono numeri complessi). Il quadrato del modulo è la probabilità di individuare un particolare stato, quando effettuiamo una misurazione.

- **Decoerenza.**

Quando si effettua una misurazione il sistema si *perturba* quindi collassa solo in uno degli stati che prima erano sovrapposti. Si lascia quindi una traccia all'atto della misurazione, la usiamo per controllare circa eventuali intromissioni nella trasmissione

- **No-cloning.**

Impossibilità di fare una copia di un sistema quantistico. Per copiare bisogna misurare, ma misurare significa modificare il sistema e quindi lasciare una traccia.

No osservazione, no copia

- **Entanglement.**

La possibilità che due sistemi creati con una correlazione tra loro continuino a mantenere la correlazione anche se portati a grandi distanze l'uno dall'altro (al contrario del principio di località della fisica classica). Quindi una misura eseguita su uno dei due influenza anche lo stato dell'altro seppur a larghissima distanza.

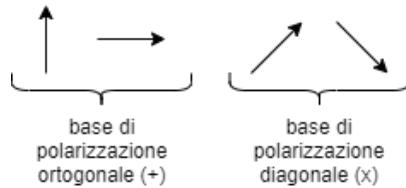
Esempio del fratello del prof. Luccio: *Il vostro partner va in Australia, se il vostro partner vi tradisce in quello stesso momento diventate cornuti (pur essendo a migliaia di chilometri di distanza).*

24.2 Protocollo BB84

Nasce nel 1984 da Bennet e Brassard. Si fa tramite lo scambio di *foton polarizzati* (particella elementare che costituisce la luce corpuscolare). Ogni fotone ha varie proprietà:

- non ha massa;
- si muove alla velocità della luce;
- ha una sua *polarizzazione* (piano di oscillazione del suo campo elettrico).

Gli stati di polarizzazione da noi considerati sono quattro, che per comodità raggruppiamo in due *basi di polarizzazione*.

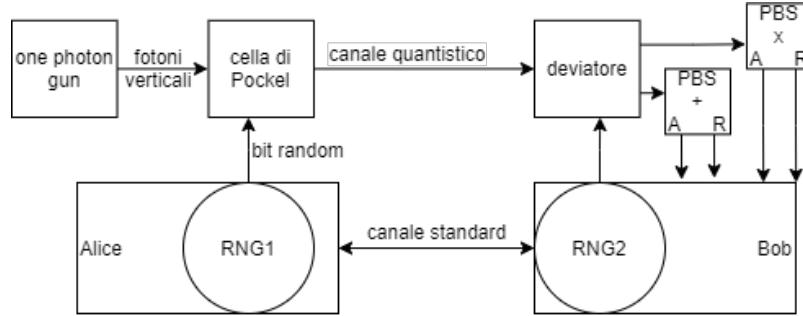


La direzione di polarizzazione può essere qualsiasi quindi scegliamo una base per rappresentarla (come se fosse un vettore). In ognuno di questi stati codifichiamo un bit:

- | | |
|--------------------------|------------------|
| • verticale: 0 | • +45°: 0 |
| • orizzontale : 1 | • -45°: 1 |

Le scelte sono arbitrarie, stabilite dagli interlocutori: sanno quale base è stata adottata e compiono misurazioni secondo la base adottata. Se non si conosce quale base è stata usato non possiamo effettuare misurazioni corrette e certe ma solo probabilistiche: nel protocollo ci baseremo proprio su questa probabilità! La misura è completamente casuale perché a seguito di una misurazione andiamo ad alterare il sistema, dunque perdiamo le informazioni precedenti.

24.2.1 Premessa: struttura generale dell'hardware



Ci si muove su due canali:

- uno quantistico (fibra ottica),
- uno standard (Alice e Bob devono comunicarsi le basi che hanno utilizzato per misurare, vedremo bene più avanti).

Lato Alice Supponiamo che Alice voglia trasmettere un messaggio a Bob.

- **One Photon Gun:** emette singoli fotoni, tutti con polarizzazione verticale.
- **Cella di Pockel:** impone una certa polarizzazione. Questa è scelta da un *Random Number Generator*, cioè Alice sceglie casualmente la polarizzazione del fotone.

Lato Bob Bob riceve i fotoni, ma non sa cosa vuole trasmettere Alice. Non ha presente neanche la base: segue che per prima cosa dovrà scegliere come misurare il fotone, lo fa in modo casuale attraverso un *Random Number Generator*.

- **SW:** *deviatore*, che si comporta in base al valore scelto da RNG2. In base al valore scelto cambia lo strumento con cui misuriamo il fotone.
- **PBS** (*Beam splitter polarizzante*): ci dà la corretta polarizzazione del fotone **se e solo se** la base di generazione e la base di misurazione sono le stesse utilizzate da Alice al momento dell'invio del fotone.

24.2.1.1 Beam Splitter Polarizzante (PBS)

Il Beam Splitter polarizzante presenta un suo asse di polarizzazione S , in contrasto con la polarizzazione F del fotone arrivato, mentre θ è l'angolo tra le due direzioni di polarizzazione. Misura il fotone tramite la deviazione verso una delle due uscite:

- A (assorbimento)
- R (riflessione)

La polarizzazione uscente non è quella del fotone in ingresso, ma una delle due polarizzazioni del PBS: S o S ortogonale. Da cosa dipende l'uscita? Dall'angolo θ . Affermiamo che il fotone:

- esce da A (polarizzazione S) con probabilità $\cos^2 \theta$
- esce da R (polarizzazione S ortogonale) con probabilità $\sin^2 \theta$

Esempi in cui la polarizzazione è mantenuta Può succedere che la polarizzazione uscente sia la stessa del fotone entrante. Sia F , ribadiamo, la polarizzazione del fotone entrante.

- Se $\theta = 0$ abbiamo $F = S$, il fotone ha la stessa polarizzazione dello strumento. Le probabilità sono:

$$-\sin^2 \theta = 0 \text{ che esca da } R; \quad -\cos^2 \theta = 1 \text{ che esca da } A.$$

Il fotone uscirà sicuramente da A , con polarizzazione S .

- Se $\theta = 90$ allora $F \perp S$. Abbiamo come probabilità:

$$-\sin^2 \theta = 1 \text{ che esca da } R; \quad -\cos^2 \theta = 0 \text{ che esca da } A.$$

Il fotone uscirà sicuramente da R e avrà come polarizzazione S ortogonale. Anche in questo caso si conserva.

Distruzione dello stato precedente Se $\theta = \pm 45^\circ$ le basi di misurazione e di generazione sono diverse. Abbiamo come probabilità:

$$\cos^2 \theta = \sin^2 \theta = \frac{1}{2}$$

Il fotone ha pari probabilità di uscire da A o da R. La lettura attraverso il PBS potrebbe quindi distruggere lo stato quantistico precedente. Ricapitolando

	0 ↑	0 ↗	1 →	1 ↘
+	↑	↑→	→	↑→
x	↗↘	↗	↗↘	↘

Le colonne indicano **bit e fotone inviato da Alice**, le righe **le basi di Bob**. Si ribadisce quanto già detto: se la base coincide si ottiene lo stesso stato, se la base non coincide allora lo stato precedente viene distrutto (i possibili stati sono equiprobabili).

24.2.2 Step del protocollo

Andiamo dunque al protocollo:

1. Alice invia una sequenza S_A sul canale quantistico.
Si segna le basi usate per generare i fotoni.
2. Bob interpreta S_A con le basi scelte casualmente.
Si segna quali basi ha usato per effettuare le misurazioni dei fotoni.

3. Bob comunica ad Alice le sue basi sul canale standard

4. Alice risponde dicendo quali basi sono comuni alle sue.

Dove le basi sono concordi si prendono i bit, dove le basi sono discordi si buttano.

Si ha circa il 50% di match. Alice e Bob, a seguito del confronto e in assenza di interferenze di Eve, possiedono $S'_A = S'_B$ sottosequenze identiche formate dai bit codificati dal mittente e decodificati dal destinatario con basi comuni:

$$|S'_A| = |S'_B| = \frac{|S_A|}{2}$$

Variante al protocollo Esiste un altro algoritmo quantistico di scambio di chiavi che si basa sull'entanglement: se i due estremi misurano con la stessa base fotoni correlati sono correlati anche i loro risultati quindi A e B misurano chiavi complementari.

24.2.3 Esempio

Consideriamo il seguente esempio, dove Alice trasmette una sequenza a Bob

$S_A:$	1 0 1 1 1 0 0 ...
Base adottata da alice:	+ x + x x + x ...
Fotoni inviati da Alice:	→ ↗ → ↘ ↘ ↑ ↗ ...
Base adottata da Bob:	+ x + + + x x ...
Lettura di Bob:	→ ↗ → ↑ 50% → 50% ↘ 50% ↗ ...
$S_B:$	1 0 1 0 1 1 0 ...

Le basi concordi sono 1, 2, 3, 7 che portano alla sequenza:

$$S'_A = S'_B = 1010$$

Le riflessioni sarebbero già finite in assenza di crittoanalisti...

24.2.4 Presenza del crittoanalista ed errori sperimentalni

La questione fondamentale è che la presenza del crittoanalista altera lo stato di polarizzazione dei fotoni (ricordare la proprietà della *decoerenza*).

- Se c'è un crittoanalista esso si troverà nella stessa situazione di Bob, quindi deve scegliere le sue basi.
- Le basi scelte da Eve saranno indipendenti sia da quelle di Alice che da quelle di Bob. Se la base scelta da Bob coincide con quella di Alice allora non cambia nulla, se non coincide altera irreparabilmente il fotone!
- **Soluzione.** Alice e Bob fanno una verifica sacrificando un pezzo di S'_A e S'_B : si scambia una porzione delle chiavi comuni in posizioni prestabilite comunicandole sul canale standard. A quel punto:

- se le due sequenze sono diverse la comunicazione viene interrotta;
- altrimenti usano la porzione rimanente come chiave o per costruire la chiave.

Riprendiamo l'esempio precedente e vediamo cosa succede con Eve nel mezzo:

$S_A:$	1	0	1	1	1	0	0	...
Base adottata da Alice:	+	x	+	x	x	+	x	...
Fotoni inviati da Alice:	→	↗	→	↘	↘	↑	↗	...
Base adottata da Eve:	+	+	+	x	+	x	+	...
Lettura di Eve:	→	→ 50%	→	↘	↑ 50%	↗ 50%	→ 50%	...
$S_E:$	1	1	1	1	0	0	1	...
Base adottata da Bob:	+	x	+	+	+	x	x	...
Lettura di Bob:	→	↘ 50%	→	↑ 50%	↑	↗	↘ 50%	...
$S_B:$	1	1	1	0	0	0	1	...

Prendiamo i valori con basi concorde:

$$S'_A = 1010 \neq 1111 = S'_B$$

- Supponiamo quindi che si sacrifichi il secondo bit: $0 \neq 1$ e quindi ci si accorge di una intrusione.
- I bit non perturbati sono quelli per cui le 3 basi coincidono: $\frac{1}{4}$ delle volte. Quindi se Eve interviene: circa la metà di S'_A/S'_B sarà differente.
- La verifica dell'intrusione è quindi importantissima, permette di sapere di eventuali intrusioni e quindi evitare di perdere informazioni.

Si osservi che le variazioni nello stato di polarizzazione non dipendono solo dall'intrusione di Eve: possiamo avere errori generici in fase di trasmissione dei fotoni, delle letture, ecc...

- Si stabilisce un **quantum bit error rate** (QBER): una percentuale prevedibile di bit errati (dovuti a errori dell'apparato sperimentale).
- Si confrontano S'_A e S'_B :
 - se il numero di errore è $> \text{QBER}$ allora c'è stata una intromissione (e interrompiamo la comunicazione);
 - altrimenti sono solo errori sperimentali e si correggono tramite correttori di errori.
- Eve potrebbe intercettare pochi fotoni e quindi farsi passare per errori dell'apparato pur conoscendo alcuni bit della chiave. Proprio per questo motivo anziché usare la sequenza scambiata si usa farne l'hash ed usare quello come chiave, in questo modo se cambiano anche solo pochi bit l'hash cambia enormemente.
- Le comunicazioni su canale standard possono anche essere in chiaro, è bene tuttavia che siano autenticate, magari tramite MAC (simmetrico con chiave scambiata in anticipo).

Capitolo 25

Bitcoin

25.1 Introduzione

Il Bitcoin non è propriamente una moneta, ma un sistema di pagamento. Sfrutta i protocolli crittografici per due aspetti:

- generare nuova moneta
- attestare il possesso della valuta da parte degli utenti.

Nel compiere una transazione è necessario indicare una chiave privata: è l'unica cosa che permette di accertare il possesso dei Bitcoin, perdere la chiave significa perdere i Bitcoin.

Nascita Nasce nel 2008 con la pubblicazione di *Bitcoin: a peer to peer Electronic cash system* a cura di *Satoshi Nakamoto* (pseudonimo per una persona o un gruppo ancora ignoto).

- Nel 2009 viene rilasciato il primo software per partecipare (bitcoin-core) ed il 3 Gennaio 2009 viene creato il *blocco genesi*.
- Il 12 Gennaio 2009 avviene la prima transazione: Satoshi Nakamoto invia 10 Bitcoin ad Hal Finney (il creatore del proof of work).
- Nel 2010 avviene la prima transazione commerciale: vengono pagate due pizze.

Caratteristiche

- Questo sistema distribuito è caratterizzato da un libro contabile (*ledger*) pubblico e distribuito costituito da singoli blocchi concatenati tra di loro: la **blockchain**.
- La **blockchain** è una lista di blocchi contenenti centinaia di transazioni: si parla di catena perchè ogni blocco è legato ai precedenti tramite il calcolo di una funzione hash.
- **Il primo blocco.** Il blocco genesi è il primo blocco di questa catena: con questo si sono creati i primi 50 bitcoin.

- **Numero di bitcoin generati con la creazione di un nuovo blocco.**

Ogni volta che si riesce ad aggiungere un nuovo blocco si creano nuovi bitcoin ed ogni 4 anni circa si dimezza la moneta generata (inizialmente 50 bitcoin per blocco, ora 6.25 per blocco).

- **Deadline per la creazione di Bitcoin.**

La generazione ha un tempo massimo infatti nel 2140 l'aggiunta di blocchi non genererà più nuova moneta.

25.2 Funzionamento

Dopo aver scaricato il software necessario l'utente genera una sua coppia di chiavi

$$< K_A[\text{pub}], K_A[\text{priv}] >$$

- **Chiave pubblica** $K_A[\text{pub}]$: costituisce l'indirizzo identificativo dell'utente, si usa per ricevere Bitcoin e per verificare la firma
- **Chiave privata** $K_A[\text{priv}]$: si usa per firmare le transazioni (quindi per spendere)

Il tutto si basa sulla **crittografia a curve ellittiche**.

Attenzione Nel caso in cui venga persa la chiave privata non potremo più porci come legittimi proprietari del Bitcoin. Chiunque individui la chiave privata può porsi in modo legittimo come proprietario.

Wallet Il *wallet* è l'insieme delle credenziali che attestano la proprietà dei BTC: la coppia indirizzo/chiave privata. Ogni utente, nel software di gestione, ha un suo Wallet.

25.2.1 Transazione

La transazione è lo scambio di valuta tra due utenti. Immaginiamo che Alice voglia inviare un numero di x BTC a Bob, il messaggio avrà la forma m e verrà corredato da *hash* e *firma*

$$\begin{aligned} m &= \text{address}_A - x - \text{address}_B \\ h &= \text{SHA}_{256}(m) \\ f &= D(h, K_A[\text{priv}]) \end{aligned}$$

Nel compiere la transazione verrà diffusa in rete (rete peer-to-peer, comunicazione di tipo broadcast) la coppia $< m, f >$ ¹.

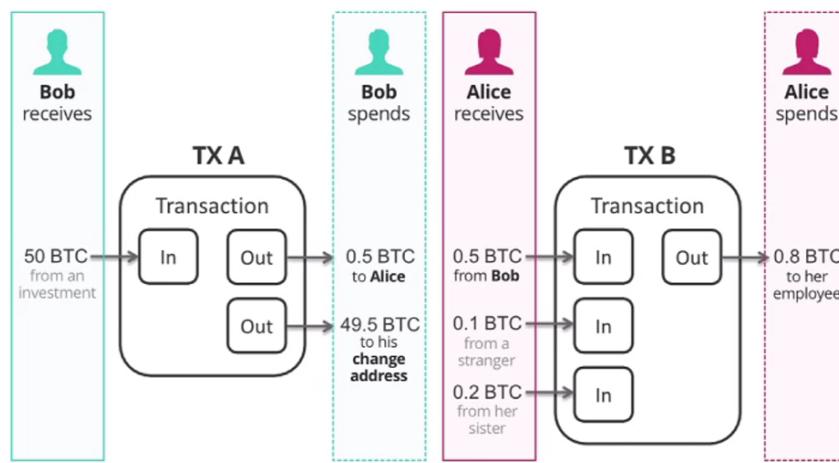
Attesa Essendo su un sistema distribuito il destinatario deve aspettare che la rete convalidi la transazione e richiede almeno 10 minuti.

¹Attenzione al refuso sul libro, si trasmette f e non h .

Molteplicità dei rami di blockchain Dopo 10 minuti il blocco verrà aggiunto alla blockchain, bisogna poi aspettare l'aggiunta di altri 6 blocchi al nostro ramo per essere sicuri al 100% di aver compiuto una transazione: questo perché sulla catena di blockchain si formano vari rami. Il ramo delle transazioni considerate effettivamente valide è solo quello ufficiale: chi compie transazioni deve stare attento su questo, altrimenti le transazioni potrebbero non essere valide (in quel caso ci si trova su *blocchi orfani*, chi ha pagato è come se non avesse pagato).

25.3 Frazionamento delle transazioni

Il frazionamento delle transazioni è questione macchinosa.



25.3.1 Primo esempio con Bob

Supponiamo di avere Bob che ha ricevuto 50 Bitcoin da un investitore: questi bitcoin sono l'input. Bob vuole spendere queste "monete", ma l'unica cosa che vuole fare inizialmente è trasmettere 0.5 Bitcoin ad Alice.

Problema Il patrimonio in mano a Bob è rappresentato da una o più transazioni, non è possibile scindere il contenuto di una transazione trasmettendo soltanto 0.5 Bitcoin e ignorando i rimanenti 49.5 Bitcoin. Se supponiamo i 50 Bitcoin come una banconota è come se Bob tagliasse con le forbici un pezzo di banconota.

Soluzione Nella transazione dobbiamo trasmettere tutti e 50 i Bitcoin, divideremo la cosa nel seguente modo:

- 0.5 Bitcoin vanno ad Alice
- 49.5 vengono trasferiti a un indirizzo in mano a Bob (in un certo senso indica di volerli spedire a se stesso, potrebbe indicare la sua stessa chiave pubblica ma di solito non si fa così²).

²Per mantenere l'anonimato

25.3.2 Secondo esempio con Alice

Successivamente Alice può spendere o meno i bitcoin ricevuti. Supponiamo che voglia trasmettere a un impiegato 0.8 Bitcoin. Può farlo unendo nell'input l'output di più transazione, precisamente:

- 0.5 Bitcoin ricevuti prima da Bob
- 0.1 Bitcoin ricevuti da uno sconosciuto (ah quindi Alice parla con gli sconosciuti)
- 0.2 Bitcoin dalla sorella

Si possono spendere **output non spesi di precedenti transazioni** a noi destinate.

25.3.3 Validazione

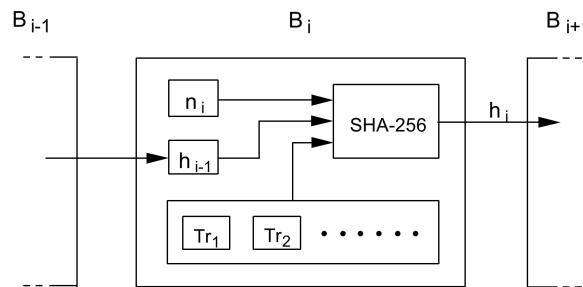
Gli utenti ricevono dal broadcast queste transazioni, le controllano (cercano nel passato se la valuta è presente nel conto di chi invia) le mettono assieme a gruppi (100, 200 anche 1000) e poi provano ad aggiungerle alla blockchain.

Come avviene l'aggiunta? Ci sono vari modi per aggiungere i blocchi alla blockchain, noi vedremo il *proof of work*.

double spending La blockchain prevede il *double spending*: per fare un paragone con la realtà è come se io pagassi due servizi diversi usando la stessa moneta, solo che al pagamento del secondo servizio i soldi non dovrebbero esserci più (sono già stati usati!). La cosa succede con la creazione di nuove transazioni nel periodo di validazione.

25.4 Struttura di un blocco della *blockchain*

Sono un nodo: arrivano le transazioni, le raggruppo in blocchi e cerco di validarle.



- Creo un blocco che le contiene, aggiungo la transazione finale che redirige verso di me destinatario la ricompensa.
- Nel blocco c'è h_{i-1} cioè l'hash del blocco precedente nella catena: è importante perchè stabilisce la concatenazione col blocco precedente
- Nel blocco c'è n_i detto *nonce*, un intero.

Di queste informazioni faccio l'hash SHA-256 ed ottengo h_i , che andrà in ingresso nel blocco successivo.

Richiesta

- La richiesta è quella di ottenere un $h_i <$ di una certa soglia fissata dal sistema.
- Come faccio? Con una ricerca enumerativa: trovare quel valore intero n_i (il *nonce*) tale da soddisfare la richiesta. La richiesta è soddisfatta se troviamo un h_i con T zeri all'inizio.
- Il parametro T è fissato dal sistema e varia in base alla potenza della rete. Questo è scelto in modo da portare il tempo di validazione di un blocco a circa 10 minuti. In media ci vogliono 2^T tentativi.

25.5 Miner e mining

Chi cerca di attaccare un nuovo blocco individuando il *nonce* è detto **miner**. I miner sono i nodi che validano le transazioni aggiungendo nuovi blocchi alla blockchain. Si parla di **validazione tramite mining**.

La *proof of work* è la ricerca del *nonce*.

- Cercare il *nonce* è difficile
- Verificare il *nonce* è semplice, quindi chi lo trova lo diffonde in broadcast a tutti i nodi, gli altri lo verificano, controllano la validità delle transazioni ed esprimono il loro consenso: prendono il nuovo nodo e cercheranno di attaccare nuovi nodi ad esso.
- Se la rete ad un certo punto ha delle biforcati si tende a privilegiare i rami con più transazioni perché è più probabile che siano accettati da tutta la rete.

Mining pool Inizialmente Bitcoin doveva essere democratico e quindi tutti avrebbero potuto partecipare al mining, quello che è successo invece è stato che alcuni piccoli gruppi di utenti hanno unito le forze e messo a disposizione grande hardware per minare tutti assieme, ci si spartisce quindi lo spazio di ricerca del nonce e si dividono i profitti tra i vari partecipanti.

Aspetti sociali Il mining ormai è poco sociale in quanto lo fanno in pochi e guadagnano in pochi. Inoltre si perde un sacco di energia elettrica per calcoli "inutili". Sono state quindi sviluppate altre monete digitali che dirigono il proof work verso ambiti di ricerca per non rendere tutta questa energia elettrica sprecata

Attacchi alla blockchain Per attaccare la blockchain si deve disporre di una grande potenza di calcolo. Si possono mettere d'accordo più persone per far sì che si attaccino blocchi a piacere. Si deve disporre del 51% della rete: il che è abbastanza difficile, se avessi questa potenza però sarebbe più remunerativo giocare onestamente.

Parte VIII

Appendici

Appendice A

Nozioni di algebra modulare

A.1 Utilità dell'algebra modulare in crittografia

E' fondamentale in crittografia in quanto:

- riduce lo spazio dei numeri sui quali si opera e rende più veloci i calcoli
- rende difficili alcuni problemi computazionalmente semplici (o banali) nell'algebra standard

Nell'algebra modulare ad esempio le funzioni tendono ad essere "imprevedibili". Mettiamo a confronto due funzioni con la seguente tabella: La prima funzione appartiene all'algebra

x	1	2	3	4	5	6	7	8	9	10	11	12
2^x	2	4	8	16	32	64	128	256	512	1024	2048	4096
$2^x \bmod 13$	2	4	8	3	6	12	11	9	5	10	7	1

normale ed è monotona (presenta un andamento all'occhio non casuale), mentre la seconda all'algebra modulare è "imprevedibile" (addirittura pare casuale).

A.2 Proprietà principali dell'operatore modulo

Queste proprietà sono già state viste a Reti logiche

$$\begin{aligned}|a + b|_m &= ||a|_m + |b|_m|_m \implies (a + b) \bmod m = (a \bmod m + b \bmod m) \bmod m \\|a - b|_m &= ||a|_m - |b|_m|_m \implies (a - b) \bmod m = (a \bmod m - b \bmod m) \bmod m \\|a \cdot b|_m &= ||a|_m \cdot |b|_m|_m \implies (a \cdot b) \bmod m = (a \bmod m \cdot b \bmod m) \bmod m\end{aligned}$$

Mentre questa è una novità

$$a^{r \times s} \bmod m = (a^r \bmod m)^s \bmod m$$

A.3 Relazione di congruenza

$$a \equiv b \pmod{n}$$

Dati $a, b \geq 0$ interi e $n > 0$ diremo che a è congruo a b mod n se e solo se esiste un intero k per cui:

$$a = b + k \cdot n$$

Quindi

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

A.3.1 Differenza tra relazioni di congruenza e di uguaglianza

Nella relazione di congruenza mod n si riferisce all'intera relazione, nelle relazioni di uguaglianza invece si riferisce solo al membro dove appare:

$$5 \equiv 8 \pmod{3} \implies 5 \pmod{3} = 8 \pmod{3}$$

$$5 \neq 8 \pmod{3} \implies 5 \neq 2$$

$$2 = 8 \pmod{3}$$

A.4 Numeri coprimi (ribadiamo, da *Wikipedia*)

Numeri primi Un intero si dice primo se divisibile solo per 1 e per se stesso.

Numeri coprimi Gli interi a e b si dicono coprimi (o primi tra loro, nella definizione di primi si parla di un solo numero, nella definizione di coprimi si mettono a confronto due numeri) se e solo se essi non hanno nessun divisore comune eccetto 1 e -1 o, in modo equivalente, se

$$\text{MCD}(a, b) = 1$$

Esempi

- I numeri 6 e 35 sono coprimi, poichè $\text{MCD}(6, 35) = 1$.
- I numeri 6 e 27 non sono coprimi, poichè entrambi divisibili anche per 3. Inoltre

$$\text{MCD}(6, 27) = 3$$

A.5 Insieme degli interi e insieme degli interi coprimi con n

Dato $n \in \mathbb{N}$ definiamo:

- L'insieme dei numeri interi da 0 a $n - 1$

$$Z_n = \{0, 1, 2, \dots, n - 1\}$$

- Un sottoinsieme degli elementi che contiene solo gli elementi coprimi con n

$$Z_n \supseteq Z_n^* = \{\text{elementi di } Z_n \text{ coprimi con } n\}$$

- Se n è primo: $Z_n^* = \{1, 2, 3, \dots, n - 1\}$
- Se n non è primo Z_n^* è computazionalmente difficile da calcolare.

A.6 Funzione di Eulero

La funzione di Eulero restituisce la cardinalità dell'insieme dei numeri $\in Z_n$ coprimi ad n (Z_n^*)

$$\phi(n) = |Z_n^*|$$

- Se n primo $\implies \phi(n) = n - 1$ (immediato, visto che il numero è primo e in quel caso tutti gli elementi di Z_n tranne lo zero appartengono a Z_n^*)
- Se n è composto la cosa è un po' più complicata

$$\phi(n) = n \cdot \left(1 - \frac{1}{p_1}\right) \cdot \dots \cdot \left(1 - \frac{1}{p_k}\right)$$

con p_1, \dots, p_k i fattori primi di n presi senza molteplicità.

Nel caso di $n = p \cdot q$ con p e q primi (quindi n semiprimo):

$$\phi(n) = |Z_n^*| = (p - 1) \cdot (q - 1)$$

A.7 Teorema di Eulero

$$\boxed{\forall n > 1, \forall a \in Z_n^*: a^{\phi(n)} \equiv 1 \pmod{n}}$$

Prendiamo ad esempio l'insieme Z_5^*

$$\begin{aligned} Z_5 &= \{0, 1, 2, 3, 4\} \\ Z_5^* &= \{1, 2, 3, 4\} \end{aligned}$$

otteniamo

$$\begin{aligned} 3^4 &\equiv 1 \pmod{5} \\ 3^4 \pmod{5} &= 1 \pmod{5} \\ 81 \pmod{5} &= 1 \pmod{5} \implies 1 = 1 \end{aligned}$$

Piccolo teorema di Fermat (corollario del precedente) $\forall n$ primo e $\forall a \in Z_n^*$:

$$a^{n-1} \equiv 1 \pmod{n}$$

La cosa è abbastanza scontata: abbiamo detto che se n è primo allora $\phi(n) = n - 1$, dunque otteniamo il corollario semplicemente sostituendo $\phi(n)$.

A.7.1 Conseguenze dei risultati: formula per trovare l'inverso

$\forall a \in Z_n^*$ abbiamo

$$\begin{aligned} [a^{\phi(n)} = a \cdot a^{\phi(n)-1}] &\longrightarrow a \cdot a^{\phi(n)-1} \equiv 1 \pmod{n} \\ a \cdot (a^{\phi(n)-1} \pmod{n}) &= 1 \pmod{n} \end{aligned}$$

Sapendo che l'inverso del numero x è un numero y tale che $xy = 1$ poniamo

$$a \cdot a^{-1} \equiv 1 \pmod{n}$$

cioè

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

Quindi $a^{-1} \pmod{n}$ si può calcolare se si conosce la funzione di Eulero $\phi(n)$, quindi se si conoscono p, q, \dots primi fattori di n .

A.8 Teorema sull'inverso: ammissione e numero di soluzioni

L'equazione

$$a \cdot x \equiv b \pmod{n}$$

ammette soluzioni se e solo se $\text{MCD}(a, n)$ divide b . In questo caso si hanno esattamente $\text{MCD}(a, n)$ soluzioni distinte.

A.8.1 Corollario: unicità della soluzione

L'equazione precedente

$$a \cdot x \equiv b \pmod{n}$$

ammette un'unica soluzione se e solo se a, n sono coprimi.

Inverso con $b = 1$ L'inverso è calcolabile con la seguente formula, vista poco indietro

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

ma per conoscere $\phi(n)$ mi serve fattorizzare, **e questo è un problema difficile**. La buona notizia è che calcolare l'inverso non è sempre difficile, ricorriamo all'Algoritmo di Euclide.

A.9 Recap delle nozioni affrontate fino ad ora

- Abbiamo visto il concetto di numeri coprimi.
- Abbiamo introdotto l'insieme degli interi coprimi rispetto a un numero n , accorgendoci delle difficoltà nel calcolare l'insieme qualora n non sia numero primo. Se invece n è primo la cosa è semplice.
- Abbiamo introdotto la funzione di Eulero, che restituisce la cardinalità dell'insieme detto al punto precedente. Permane la stessa difficoltà.
- Abbiamo introdotto il Teorema di Eulero

$$\forall n > 1, \forall a \in Z_n^*: a^{\phi(n)} \equiv 1 \pmod{n}$$

e il piccolo teorema di Fermat (corollario del precedente), dove con n primo si può dire

$$a^{n-1} \equiv 1 \pmod{n}$$

otteniamo infine una formula per calcolare l'inverso

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

- Col teorema sull'inverso abbiamo visto che sono ammesse soluzioni nella seguente formula

$$a \cdot x \equiv b \pmod{n}$$

solo se $\text{MCD}(a, n)$ divide b , e che MCD consiste nel numero di soluzioni. Con un corollario abbiamo visto che l'unicità della soluzione si ha solo con a, n coprimi, in quel caso la formula per trovare l'inverso è quella calcolata grazie al teorema di Eulero.

A.10 Algoritmo di Euclide esteso

L'algoritmo di Euclide, visto al primo anno, permette di individuare il massimo comune divisore. Nella versione estesa Euclide non restituisce solo il MCD, ma permette di trovare i valori x, y che risolvono la seguente equazione

$$a \cdot x + b \cdot y = \text{MCD}(a, b)$$

Prende in input il dividendo a , il divisore b e restituisce una terna di valori $\langle d, x, y \rangle$:

- d è $\text{MCD}(a, b)$;
- x e y sono i valori in grado di soddisfare l'equazione appena detta.

```

Euclide_esteso(a,b):
if(b == 0)  return <a,1,0>

<d', x', y'> = Euclide_esteso(b, a mod b)
<d, x, y> = <d', y', x' - floor(a/b) * y'>
return <d, x, y>

```

- **Caso base.** Se il divisore b è uguale a zero, cioè quando abbiamo

$$a \cdot x = \text{MCD}(a, 0)$$

In questo caso, con $b = 0$, abbiamo $\text{MCD}(a, 0) = a$. Segue $x = 1$ e il valore restituito col caso base.

- Se non ho subito la chiusura procedo ricorsivamente. Semplifico l'istanza di input: passo come parametri b e $a \bmod b$.
- Una volta risolto il problema dello step precedente si ottiene la terna $\langle d', x', y' \rangle$. MCD non cambia poichè

$$\text{MCD}(a, b) = \text{MCD}(b, a \bmod b)$$

Scambiamo x ed y di posizione, inoltre il terzo parametro viene ridotto ulteriormente.

$$\langle d, x, y \rangle = \left\langle d', y', x' - \left\lfloor \frac{a}{b} \right\rfloor \cdot y' \right\rangle$$

- Immaginatevi la cosa come un subacqueo che si immerge e arriva sul fondale: l'acqua sono tutte le chiamate ricorsive.
 - Quando arriva sul fondale ha calcolato il MCD (il valore d).
 - Dopo ritorna in superficie, e durante la risalita calcola i valori $\langle x, y \rangle$ che risolvono l'equazione, in primis x che nel nostro caso è l'inverso moltiplicativo.

A.10.1 Calcolo dell'inverso con l'algoritmo

Passo dalla congruenza alla formula risolta con Euclide

Possiamo usare quest'algoritmo per il calcolo dell'inverso. Se questa uguaglianza è valida

$$a \cdot x \equiv 1 \pmod{n}$$

allora esiste un valore di z tale che

$$a \cdot x = n \cdot z + 1$$

spostiamo nz nel primo membro ponendo $y = -z$ ed $n = b$

$$a \cdot x + b \cdot y = 1 \longrightarrow a \cdot x + b \cdot y = \text{MCD}(a, b)$$

a e b sono coprimi, quindi $\text{MCD}(a, b) = 1$. Usando l'algoritmo si ottengono $\langle d, x, y \rangle$, dove x è proprio il nostro inverso. Logico che il MCD, rappresentato dal valore d , sarà sempre uguale ad 1.

Esempio Vogliamo calcolare l'inverso di 5 modulo 132 riscrivo l'equazione nella forma vista e applico Euclide esteso:

$$x = 5^{-1} \bmod 132 \longrightarrow 5x + 132y = 1$$

Per prima cosa determiniamo tutte le chiamate ricorsive

$a = 5$	$b = 132$	Non abbiamo ancora $b \neq 0$
$a = 132$	$b = 5 \bmod 132 = 5$	Non abbiamo ancora $b \neq 0$
$a = 5$	$b = 132 \bmod 5 = 2$	Non abbiamo ancora $b \neq 0$
$a = 2$	$b = 5 \bmod 2 = 1$	Non abbiamo ancora $b \neq 0$
$a = 1$	$b = 2 \bmod 1 = 0$	Abbiamo $b = 0$, ci fermiamo qua

A questo punto ritorniamo su:

- L'ultimo è il caso base: $a = 1, b = 0$. Abbiamo già detto che $\text{MCD}(a, 0) = a$, quindi il MCD è 1. Restituisco $\langle a, 1, b \rangle$, quindi $\langle 1, 1, 0 \rangle$.
- Saliamo di posizione e applichiamo le formule viste

$$\begin{array}{lll} \langle a = 2, b = 1 \rangle & \langle d' = 1, x' = 1, y' = 0 \rangle & \left\langle d = 1, x = 0, y = 1 - \left\lfloor \frac{2}{1} \right\rfloor \cdot 0 = 1 \right\rangle \\ \langle a = 5, b = 2 \rangle & \langle d' = 1, x' = 0, y' = 1 \rangle & \left\langle d = 1, x = 1, y = 0 - \left\lfloor \frac{5}{2} \right\rfloor \cdot 1 = -2 \right\rangle \\ \langle a = 132, b = 5 \rangle & \langle d' = 1, x' = 1, y' = -2 \rangle & \left\langle d = 1, x = -2, y = 1 - \left\lfloor \frac{132}{5} \right\rfloor \cdot (-2) = 53 \right\rangle \\ \langle a = 5, b = 132 \rangle & \langle d' = 1, x' = -2, y' = 53 \rangle & \langle d = 1, x' = 53, y' = [\text{Non ci interessa}] \rangle \end{array}$$

- Il MCD è $d = 1$, mentre l'inverso è $x = 53$. Tutto trovato in tempo polinomiale.

A.11 Algoritmo delle quadrature successive (o esponenziazione veloce)

Abbiamo incontrato per la prima volta col test di primalità di Miller-Rabin la necessità di calcolare potenze elevate (nell'algebra modulare). Col seguente algoritmo saremo in grado di ridurre il numero di moltiplicazioni necessarie.

$$x = y^z \bmod s$$

1. Scrivo l'esponente z come una somma di potenze di 2, con t che vale $t = \lfloor \log_2 z \rfloor = \theta(\log z)$.

$$z = \sum_{i=0}^t k_i \cdot 2^i \quad k_i \in \{0, 1\}$$

2. Calcolo tutte le operazioni

$$y^{2^i} \bmod S \quad 1 \leq i \leq t = \lceil \log_2 z \rceil$$

ciascuna si calcola come quadrato della precedente.

$$y^{2^i} \bmod S = (y^{2^{i-1}})^2 \bmod S$$

3. Calcoliamo $x = y^z \bmod S$ col seguente prodotto

$$x = \prod_{i:k_i \neq 0} y^{2^i} \bmod S$$

dove la condizione esclude tutte le potenze di 2 che non hanno influito sulla scomposizione dell'esponente z .

A livello di costo abbiamo al più $O(t) = \theta(\log_2 z)$ quadrature, quindi $O(\log z)$ moltiplicazioni. Il numero di moltiplicazioni cresce come $\log z$, ciascuna moltiplicazione ha un costo che va col quadrato del numero di cifre: otteniamo un algoritmo complessivamente cubico.

Algoritmo polinomiale nella dimensione dei dati

Esempio Vogliamo calcolare $x = 9^{45} \bmod 11$, dove $z = 45$

1. Scomponiamo $z = 45 = 2^5 + 2^3 + 2^2 + 2^0 = 32 + 8 + 4 + 1$
2. Calcoliamo le $k = \lfloor \log_2 45 \rfloor = 5$ operazioni

	$9^1 \bmod 11 = 9 \quad (\text{Bonus})$
1.	$9^2 \bmod 11 = (9)^2 \bmod 11 = 4$
2.	$9^4 \bmod 11 = (4)^2 \bmod 11 = 5$
3.	$9^8 \bmod 11 = (5)^2 \bmod 11 = 3$
4.	$9^{16} \bmod 11 = (3)^2 \bmod 11 = 9$
5.	$9^{32} \bmod 11 = (9)^2 \bmod 11 = 4$

3. Selezioniamo quelle che intervengono nella scomposizione dell'esponente: (6), (4), (3), (1). A questo punto possiamo fare i calcoli

$$\begin{aligned} y^z \bmod 5 &= 9^{45} \bmod 11 = 9^{32+8+4+1} \bmod 11 = 9^{32}9^89^49^1 \bmod 11 = \\ &= ((9^{32} \bmod 11)(9^8 \bmod 11)(9^4 \bmod 11)(9^1 \bmod 11)) \bmod 11 = \\ &= (4 * 3 * 5 * 9) \bmod 11 = 1 \end{aligned}$$

Si ricordi da Reti logiche la seguente proprietà dell'operatore modulo

$$|x \cdot y|_\alpha = ||x|_\alpha \cdot |y|_\alpha|_\alpha$$

La proprietà può essere generalizzata applicandola a un prodotto di n elementi.

Appendice B

Funzione *floor* e *ceil*

floor La funzione $\text{floor}(x)$ restituisce il più grande intero minore o uguale ad x

$$\text{floor}(x) = \lfloor x \rfloor \stackrel{\text{def}}{=} \max\{n \in \mathbb{Z} | n \leq x\}$$

ceil La funzione $\text{ceil}(x)$ restituisce il più piccolo intero maggiore o uguale a x

$$\text{ceil}(x) = \text{ceiling}(x) = \lceil x \rceil \stackrel{\text{def}}{=} \min\{n \in \mathbb{Z} | n \geq x\}$$

Esempi a confronto

floor

$$\begin{aligned}\lfloor 3 \rfloor &= 3 \\ \lfloor 2.6 \rfloor &= 2 \\ \lfloor -2.6 \rfloor &= -3\end{aligned}$$

ceil

$$\begin{aligned}\lceil 3 \rceil &= 3 \\ \lceil 2.6 \rceil &= 3 \\ \lceil -2.6 \rceil &= -3\end{aligned}$$

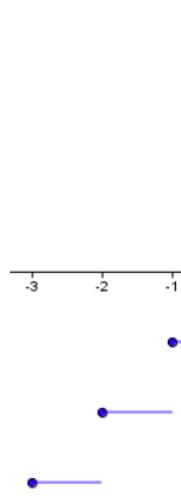


Grafico della funzione *floor*

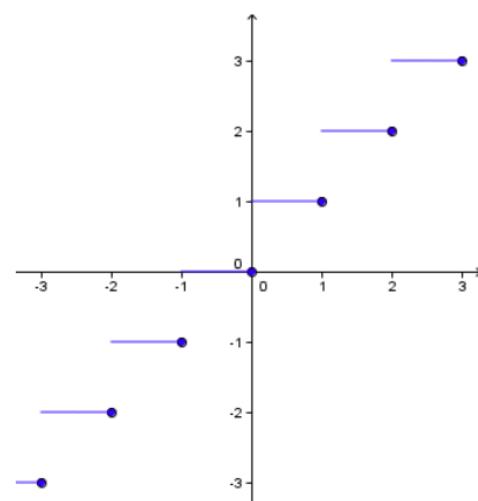


Grafico della funzione *ceil*

http://www.batmath.it/interattive/ggb_tutorials/floor_ceil/floor_ceil.htm

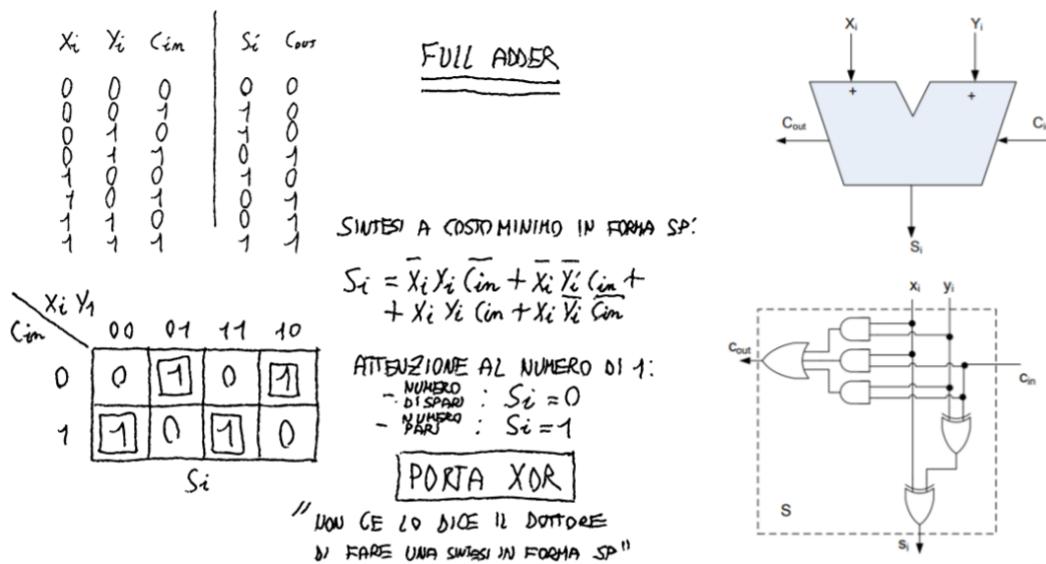
Appendice C

Operatore XOR

Molto spesso la professoressa utilizza come sinonimo di XOR la frase *somma in modulo 2*. La cosa torna se andiamo a rivedere alcune cose viste a Reti logiche.

Full adder in base 2

A Reti logiche abbiamo visto le caratteristiche dell'addizione nell'insieme dei numeri naturali. In particolare, abbiamo fatto la sintesi del *full adder* in base 2, cioè un sommatore a una cifra. Al di là del *carry* C_{out} dobbiamo ricordarci come viene ottenuto S_i .



S_i è ottenuto a partire da una porta XOR a tre ingressi (dove si considera anche C_{in}).

Elemento neutro ed elemento non neutro dello XOR

- **Zero elemento neutro:** $x \oplus 0 = x$, cioè $1 \oplus 0 = 1, 0 \oplus 0 = 0$
- **Uno elemento non neutro:** $x \oplus 1 = \bar{x}$, cioè $0 \oplus 1 = 1, 1 \oplus 1 = 0$

Appendice D

Esercizi (raccolta incompleta)

Esercizio d'esame.

Un sistema crittografico impiega chiavi private di 46 bit. Per decifrare un messaggio m data la chiave, un programma in assembler impiega un ciclo di 128 istruzioni ripetuto in media tante volte quanti sono i bit che costituiscono m . Impiegando un calcolatore che esegue un'operazione assembler in un tempo medio di 10 n s , indicare in ordine di grandezza quanti anni sarebbero necessari in media per condurre un attacco esaustivo sulle chiavi per un messaggio m di 1000 bit. Indicare i calcoli eseguiti.

Risoluzione La risoluzione passa dal moltiplicare tutti i dati presenti nella consegna: il numero di possibili combinazioni di chiavi (2^{46}): il numero di bit del messaggio in chiaro (1000), il tempo di esecuzione di un'istruzione Assembler (10^{-9} s), il numero di istruzioni che caratterizzano il ciclo di ogni singolo bit del messaggio in chiaro.

$$1000 \cdot 128 \cdot (10 \cdot 10^{-9}) \cdot 2^{46} = 10^3 \cdot 2^7 \cdot 10^{-8} \cdot 2^{46} = 10^{-5} \cdot 2^{53} = 9 \cdot 10^{10} \text{ s}$$

concludiamo dividendo per il numero di secondi in un anno

$$\frac{9 \cdot 10^{10} \text{ s}}{60 \cdot 60 \cdot 24 \cdot 365 \text{ s}} \approx 2800 \text{ anni}$$

Esercizio d'esame.

Si vuole generare una sequenza di bit pseudocasuali utilizzando il generatore BBS basato sulla legge:

$$x(i) = [x(i-1)]^2 \bmod n \quad b(i) = 1 \Leftrightarrow "x(m-i) \text{ è dispari}"$$

1. Scegliere $n = 11 \cdot 23$ e verificare che 11 e 23 soddisfino i requisiti richiesti dal generatore BBS.

2. Sia M il proprio numero di matricola. Porre $y = M \bmod 100$ e $x(0) = y^2 \bmod n$, e indicare una sequenza di 10 bit generati, riportando i calcoli eseguiti.
3. Discutere se il generatore può considerarsi crittograficamente sicuro.

Risoluzione

1. Abbiamo visto poco fa quali sono le condizioni con cui andiamo a determinare i fattori primi nel prodotto $n = p \cdot q$. Verifichiamo supponendo $p = 11$ e $q = 23$:

- $11 \bmod 4 = 3$, verificato
- $23 \bmod 4 = 3$, verificato
- $\text{MCD}\left(2\lfloor\frac{11}{4}\rfloor + 1, 2\lfloor\frac{23}{4}\rfloor + 1\right) = \text{MCD}(5, 11) = 1$, verificato

tutte le condizioni richieste da BBS sono verificate.

2. Pongo il mio numero di matricola (*solo nella mia testa, non si dice il numero di matricola agli sconosciuti*) e svolgo l'operazione col modulo

$$y \bmod 100 = 11$$

procediamo svolgendo tutti i vari calcoli richiesti dall'algoritmo BBS. Per prima cosa il seme

$$x_0 = x(0) = 11^2 \bmod 253 = 110$$

procediamo calcolando gli altri valori (dobbiamo ottenere altri 9 valori)

$$\begin{aligned} x_1 &= (x_0)^2 \bmod 253 = 110^2 \bmod 253 = 209 \\ x_2 &= (x_1)^2 \bmod 253 = 209^2 \bmod 253 = 165 \\ x_3 &= (x_2)^2 \bmod 253 = (165)^2 \bmod 253 = 154 \\ x_4 &= (x_3)^2 \bmod 253 = (154)^2 \bmod 253 = 187 \\ x_5 &= (x_4)^2 \bmod 253 = (187)^2 \bmod 253 = 55 \\ x_6 &= (x_5)^2 \bmod 253 = (55)^2 \bmod 253 = 242 \\ x_7 &= (x_6)^2 \bmod 253 = (242)^2 \bmod 253 = 121 \\ x_8 &= (x_7)^2 \bmod 253 = (121)^2 \bmod 253 = 220 \\ x_9 &= (x_8)^2 \bmod 253 = (220)^2 \bmod 253 = 77 \end{aligned}$$

A questo punto concludiamo costruendo la sequenza attraverso il predicato hard-core b_i . Ci servono 10 cifre! Ricordarsi che

$$b(i) = 1 \Leftrightarrow "x(m-i) \text{ è dispari}"$$

procediamo, ponendo $m = 9$

- $b_0 = 1 \iff "77 (x_9) \text{ è dispari}" \text{ è vera}$
- $b_1 = 0 \iff "220 (x_8) \text{ è dispari}" \text{ è falsa}$
- $b_2 = 1 \iff "121 (x_7) \text{ è dispari}" \text{ è vera}$
- $b_3 = 0 \iff "242 (x_6) \text{ è dispari}" \text{ è falsa}$
- $b_4 = 1 \iff "55 (x_5) \text{ è dispari}" \text{ è vera}$
- $b_5 = 1 \iff "187 (x_4) \text{ è dispari}" \text{ è vera}$
- $b_6 = 0 \iff "154 (x_3) \text{ è dispari}" \text{ è falsa}$
- $b_7 = 1 \iff "165 (x_2) \text{ è dispari}" \text{ è vera}$
- $b_8 = 1 \iff "209 (x_1) \text{ è dispari}" \text{ è vera}$
- $b_9 = 0 \iff "110 (x_0) \text{ è dispari}" \text{ è falsa}$

La sequenza finale è

1010110110

3. Chiaramente i titoli delle sezioni precedenti forniscono già una risposta alla domanda. L'algoritmo è crittograficamente sicuro grazie all'introduzione dei cosiddetti *predicati hard-core*.

Esercizio d'esame.

Sia C una sequenza ottenuta rappresentando in binario ciascuna delle due cifre centrali del numero di matricola del candidato, prendendo per ciascuna di esse i tre bit meno significativi, concatenando questi due gruppi di bit e aggiungendo 1 in testa.

1. Eseguire l'operazione: $37^C \bmod 100$ per esponenziazioni successive indicando i calcoli eseguiti.
2. Spiegare perché tale metodo di calcolo è considerato efficiente.

Risoluzione

1. Le cifre centrali del mio numero di matricola sono 3 e 0. Le converto singolarmente in binario e le concateno: ottengo 011000. Aggiungendo in testa un uno ottengo: $c = (1011000)_2 = 1 \cdot 2^6 + 1 \cdot 2^4 + 1 \cdot 2^3 = 2^6 + 2^4 + 2^3 = 64 + 16 + 8 = 88$
2. Vogliamo calcolare $37^{88} \bmod 100$. Dai calcoli fatti precedenti abbiamo già la scomposizione dell'esponente.

3. Calcoliamo i $k = \lfloor \log_2 88 \rfloor = 6$ moduli richiesti dall'algoritmo.

$$37^1 \bmod 100 = 37 \bmod 100 = 37 \text{ (Bonus)}$$

$$37^2 \bmod 100 = 37^2 \bmod 100 = 1369 \bmod 100 = 69$$

$$37^4 \bmod 100 = 69^2 \bmod 100 = 4761 \bmod 100 = 61$$

$$37^8 \bmod 100 = 62^2 \bmod 100 = 3721 \bmod 100 = 21$$

$$37^{16} \bmod 100 = 21^2 \bmod 100 = 441 \bmod 100 = 41$$

$$37^{32} \bmod 100 = 41^2 \bmod 100 = 1681 \bmod 100 = 81$$

$$37^{64} \bmod 100 = 81^2 \bmod 100 = 6561 \bmod 100 = 61$$

4. Concludiamo

$$\begin{aligned} 37^{88} \bmod 100 &= 37^{64+16+8} \bmod 100 = 37^{64}37^{16}37^8 \bmod 100 = \\ &= [(37^{64} \bmod 100)(37^{16} \bmod 100)(37^8 \bmod 100)] \bmod 100 = \\ &= [61 \cdot 41 \cdot 21] \bmod 100 = 5251 \bmod 100 = 21 \end{aligned}$$

Esercizio da esame.

Questo esercizio ha lo scopo di dimostrare che un cifrario affine iterato ha la stessa sicurezza di un cifrario singolo. Si considerino i due cifrari affini:

- $C_1(x) = (a_1 \cdot x + b_1) \bmod 26$
- $C_2(x) = (a_2 \cdot x + b_2) \bmod 26$

Dimostrare che esiste un cifrario affine C_3 tale che $C_3(x) = C_2(C_1(x))$.

Risoluzione Nella risoluzione si ricordi le proprietà principali dell'operatore modulo (nell'apposita appendice), che ci permettono di sostituire x in C_2 senza dover pensare all'operatore modulo.

$$\begin{aligned} C_2(C_1(x)) &= (a_2 \cdot C_1(x) + b_2) \bmod 26 = (a_2 \cdot (a_1 \cdot x + b_1) + b_2) \bmod 26 \\ &= (a_2 a_1 \cdot x + a_2 b_1) \bmod 26 = (a_3 \cdot x + b_3) \bmod 26 \end{aligned}$$

Per i duri di comprendonio (io incluso) Calcoli in più per riflettere...

$$\begin{aligned} |a_2 \cdot x + b_2|_{26} &= ||a_2 \cdot x|_{26} + |b_2|_{26}|_{26} = |||a_2|_{26} \cdot |x|_{26}|_{26} + |b_2|_{26}|_{26} = \\ &= |||a_2|_{26} \cdot |a_1 \cdot x + b_1|_{26}|_{26} + |b_2|_{26}|_{26} = |a_2 \cdot (a_1 \cdot x + b_1) + b_2|_{26} \end{aligned}$$

Esercizio da esame.

Se nei cifrari affini si lavora modulo 27 invece che modulo 26, quante sono le chiavi possibili?

Risoluzione Alla base di tutti i ragionamenti sta la condizione che a e 27 devono essere coprimi, cioè

$$\text{MCD}(a, 27) = 1$$

- b è un valore $\in [0, 26]$, quindi 27 valori possibili.
- Per quanto riguarda a osserviamo dalla scomposizione che $27 = 3^3$, dunque tutti i multipli di 3 sono esclusi. Abbiamo $\phi(27) = 18$.

In conclusione

$$\text{Numero chiavi} = (18 \cdot 27) - 1$$

Dobbiamo escludere la coppia $(1, 0)$, che non altera il carattere nella cifratura (che comunque mi permette di dire $\text{MCD}(1, 27) = 1$, *si ringrazia la dispensa di Federico Matteoni per l'osservazione*).

Esercizio da esame.

Dato un cifrario affine $(\text{mod } 26)$, si fa un attacco di tipo testo in chiaro scelto (*chosen plain-text attack*) usando il testo *hahaha*. Il testo cifrato è *nonono*. Determinare la funzione di cifratura.

Risoluzione Ricordiamoci la formula del cifrario affine

$$\text{pos}(y) = (a \cdot \text{pos}(x) + b) \bmod 26$$

Conosciamo due corrispondenze

$$\begin{cases} \text{pos}(N) = 13 = (a \cdot \text{pos}(H) + b) \bmod 26 = (a \cdot 7 + b) \bmod 26 \\ \text{pos}(O) = 14 = (a \cdot \text{pos}(A) + b) \bmod 26 = b \bmod 26 = b \end{cases}$$

quindi

$$\begin{cases} 13 = (7a + b) \bmod 26 \\ 14 = b \end{cases} \implies \begin{cases} a = 11 \\ b = 14 \end{cases}$$

Esercizio da esame. Si decifri il seguente crittogramma, sapendo che $h = 8$

REXETSIH ONSICESI UCIFTFID REHTLIET

Per risolvere l'esercizio si guardi anche le cose dette più avante relative alla decifrazione. Con $h = 8$ sappiamo che abbiamo blocchi da 8 caratteri ciascuno: all'interno di ciascun blocco dobbiamo cercare di ricostruire una frase di significato compiuto. Otteniamo (qua bisogna andare ad occhio davvero, si parta sempre dal primo blocco visto che un blocco non è detto equivalga a una parola di senso compiuto)

THISEXER -> 4,7,6,5,3,2,1,0
 THISEXER CISEISNO TDIFIFICU LTEITHER

Soluzione: *THIS EXERCISE IS NOT DIFFICULT EITHER.*

Esercizio da esame.

Nel codice One-Time Pad si sostituisca l'operatore XOR con OR, o con XNOR. Spiegare, per i due casi, se il protocollo funziona con le stesse proprietà del codice originale.

Risoluzione

- In presenza dell'operatore OR non vale più la proprietà iniettiva: questo significa che a partire da più messaggi m_k è possibile ottenere, con lo stesso chiave, il crittogramma c (non è un problema se messaggi diversi corrispondono allo stesso crittogramma, con chiavi diverse). L'assenza di iniettività provoca ambiguità nella decifrazione. Si ha inoltre inferenza di conoscenza: se un bit è uguale a zero nel crittogramma lo è anche nel messaggio in chiaro.
- Lo XOR è uguale a 1 quando i bit sono diversi e 0 quando i bit sono uguali. Nello XNOR (XOR negato) la regola è invertita. Questo non rende non funzionante il One-Time Pad.

Esercizio da esame.

In un cifrario A esistono un messaggio m e un crittogramma c tali che:

$$P(M = m) = p < \frac{1}{4} \quad P(M = m | C = c) = 1 - p$$

Spiegare se A può essere un cifrario perfetto e le conseguenze per un crittoanalista per la coppia (m, c) indicata.

Risoluzione Calcolando la probabilità condizionata otteniamo $1 - \frac{1}{4} = \frac{3}{4}$. Il fatto è che le due probabilità sono diverse, e quella appena calcolata inferisce conoscenza (capiamo che è altamente probabile che il crittogramma c rappresenti il messaggio m): non abbiamo un cifrario perfetto.

Esercizio da esame.

Sia M il numero di matricola del candidato. Si converta M in una sequenza binaria B trasformando ordinatamente in binario ogni cifra decimale di M , prendendo per ciascuna di esse i tre bit meno significativi e concatenando tali gruppi di tre bit.

1. Indicare la sequenza B , proporre una chiave K di 18 bit ottenuta lanciando idealmente una moneta e trasformare B mediante One-Time Pad utilizzando K .
2. Spiegare se il cifrario può ritenersi sicuro per messaggi binari di lunghezza multipla di 18 utilizzando come chiave una ripetizione di K per il numero di volte necessario.

Risoluzione L'esercizio è una banale applicazione pratica del *One-Time Pad*. Supponiamo che il numero di matricola sia $M = 123456$: dalla concatenazione dei bit otteniamo la seguente sequenza binaria

001010011100101110

1. Nel primo applichiamo letteralmente la funzione di cifratura del One-Time Pad. Supponiamo di aver ottenuto la seguente chiave K

$$K = 011001010110011101$$

applichiamo lo XOR ottenendo così il crittogramma

$$c = 001010011100101110 \oplus 011001010110011101 = 10011001010110011$$

2. No, il cifrario non può ritenersi sicuro se utilizziamo più volte la stessa chiave. Questa cosa è fondamentale: con messaggi cifrati con la stessa chiave possiamo fare il seguente calcolo

$$c_1 \oplus c_2 = m_1 \oplus k \oplus m_2 \oplus k = m_1 \oplus m_2$$

ciò significa individuare eventuali corrispondenze, quindi non avere un cifrario perfetto.

Esercizio da esame.

1. Sia C una sequenza ottenuta rappresentando in binario ciascuna delle due cifre centrali del numero di matricola del candidato, prendendo per ciascuna di esse i tre bit meno significativi e concatenando questi due gruppi di tre bit. Nella fase i -esima del DES, C costituisca la parte iniziale della sequenza in ingresso della S-box. Indicare la sequenza C .

2. Indicare (con interi crescenti tra 1 e 32) la sequenza POS di posizioni dei bit di $D[i]$ influenzati da C spiegando come è stato ottenuto il risultato.
3. Posto che la parte sinistra $S[i-1]$ del messaggio entrante nella fase i sia una sequenza di 1, indicare il valore dei bit di $D[i]$ di cui alla domanda precedente, riportando i calcoli eseguiti.

Risoluzione Per quanto riguarda le tabelle necessarie si considerino le figure attorno a pagina 107 del libro.

1. Prendo le cifre centrali del mio numero di matricola: 30. Ottengo la seguente sequenza C

$$C = 011000$$

2. Andiamo ad applicare quanto svolto nella funzione non lineare S-box: per prima cosa passiamo dalla S-box vera e propria, che costituisce la parte non lineare della funzione, e successivamente poniamo l'output della S-box vera e propria in una rete di permutazione P .

- Prendiamo la tabella della S-box e la sequenza in ingresso. Considero gli estremi della sequenza come numero di riga e le quattro cifre interne come numero di colonna. Otteniamo

$$\text{Num. riga} = 0 \quad \text{Num.colonna} = (1100)_2 = 12$$

Dalla S-box otteniamo

$$\text{S-box}[0, 12] = (5)_{10} = (0101)_2$$

- La sequenza ottenuta va in ingresso nella rete di permutazione P . Abbiamo detto che la S-box ha manipolato i quattro bit meno significativi: osserviamo dalla tabella che
 - il bit in posizione 1 si trova in posizione 9
 - il bit in posizione 2 si trova in posizione 17
 - il bit in posizione 3 si trova in posizione 23
 - il bit in posizione 4 si trova in posizione 31
3. Applichiamo l'operatore XOR, tenendo a mente che l'altra sequenza è caratterizzata da soli 1

- $0 \oplus 1 = 1$
- $1 \oplus 1 = 0$
- $0 \oplus 1 = 1$
- $1 \oplus 1 = 0$

Esercizio da esame.

Si trasformi il sistema DES complementando tutte le uscite della S-box. Sia M il proprio numero di matricola. Si converta M in una sequenza binaria B trasformando ordinatamente in binario ogni cifra decimale di M e prendendo per ciascuna di esse i tre bit meno significativi. Si estenda B fino a contenere in totale 48 bit aggiungendovi zeri a destra: tale sequenza sia l'uscita del blocco EP del DES, con chiave $k[0] = 1010\dots10$.

1. Indicare la sequenza B.
 2. Nella fase 1 del DES determinare il valore dei primi 4 bit a sinistra in uscita dalla S-box, spiegando come è stato ottenuto il risultato.
 3. Commentare se il DES così modificato appaia o meno palesemente meno sicuro della versione standard.

Risoluzione Per quanto riguarda le tabelle necessarie si considerino le figure attorno a pagina 107 del libro.

1. Consideriamo come numero di matricole $M = (123456)_{10}$, otteniamo:

$$B = 001\ 010\ 011\ 100\ 101\ 110$$

portiamo la sequenza a 48 bit includendo gli zeri richiesti come cifre meno significative $B = 001\ 010\ 011\ 100\ 101\ 110\ 000\ 000\ 000\ 000\ 000\ 000\ 000\ 000\ 000\ 000\ 000\ 000$

2. Facciamo le stesse cose spiegate nell'esercizio precedente, in aggiunta complementiamo i bit alla fine.

- Il numero di riga è 0, il numero di colonna è $(0101)_2 = (5)_{10}$. Otteniamo

$$\text{S-box}[0, 10] = (6)_{10} = (0110)_2$$

- Per quanto riguarda la permutazione questa volta non stiamo lavorando sulle cifre meno significative, ma su quelle più significative! Osserviamo che:

- il bit in posizione 32 si trova in posizione 21
 - il bit in posizione 31 si trova in posizione 15
 - il bit in posizione 30 si trova in posizione 27
 - il bit in posizione 29 si trova in posizione 5

- Complemento i bit

$0110 \Rightarrow 1001$

3. Abbiamo visto la proprietà del DES con m, c complementati: date le coppie $\langle m, k \rangle, \langle \bar{m}, \bar{k} \rangle$ otteniamo, rispettivamente, c e \bar{c} . Si tenga conto che l'operatore XOR si applica solo a una parte del blocco, quella dove è presente la S-box: introdurre la complementazione significa ottenere, con la seconda coppia, c e non più \bar{c} .

Esercizio da esame.

La proprietà di non linearità di qualsiasi cifratura a blocchi è fondamentale per la sua sicurezza. Infatti, si supponga di avere una cifratura lineare a blocchi Cl che cifra blocchi di 128 bit di testo in chiaro in 128 bit di testo cifrato, usando una chiave k (la lunghezza di k è irrilevante). Dunque, per ogni coppia di messaggi m_1 e m_2 risulta

$$\text{Cl}(m_1 \oplus m_2, k) = \text{Cl}(m_1, k) \oplus \text{Cl}(m_2, k)$$

Descrivere come un avversario che abbia 128 testi cifrati scelti possa decifrare qualsiasi testo cifrato senza conoscere la chiave segreta k .

NOTA: “testo cifrato scelto” significa che l’avversario ha la possibilità di scegliere un testo cifrato e ottenerne la decifrazione. In questo caso si hanno 128 coppie di testo in chiaro/testo cifrato e si ha la possibilità di scegliere i testi cifrati.

Risoluzione La non linearità previene attacchi dove il crittoanalista, data una sequenza di crittogrammi scelti in modo oculato, riesce a decifrare crittogrammi senza conoscere la chiave k .

- Consideriamo il crittogramma c relativo ad un messaggio m Rappresentiamolo nel seguente modo (è a 128 bit)

$$c = c_1 c_2 \dots c_{128} \quad c = \bigoplus_{i:c_i=1} e^{(i)}$$

dove i è una sequenza caratterizzata da tutti zeri, tranne il bit i -esimo uguale ad 1. Si prenda il seguente esempio

$$C = 1011 = e^{(1)} \oplus e^{(3)} \oplus e^{(4)}$$

- Sfruttiamo quindi la proprietà delle funzioni di cifratura e decifrazione

$$m = D(c, k) = D\left(\bigoplus_{i:c_i=1} e^{(i)}, k\right) = \bigoplus_{i:c_i=1} D(e^{(i)}, k)$$

- La cosa calcolata precedente, possibile grazie alla linearità, ci porta ad affermare che è possibile decifrare qualunque messaggio se conosciamo le 128 sequenze

$$f^{(i)} = D(e^{(i)}, k) \quad i \in [1, 128]$$

Esercizio da esame.

La S-box del cifrario AES è composta da 16 blocchi a 8 ingressi e 8 uscite ciascuno. Qual è il numero totale di possibili funzioni che si potrebbero scegliere per costruire

ciascun blocco?

Risoluzione Negli appunti sulle Reti combinatorie di Reti logiche abbiamo visto il numero di tavole di verità possibili per reti ad N ingressi e un'uscita

$$\text{Numero di tavole di verità} = (2)^{2^N}$$

Abbiamo 2^8 combinazioni possibili per l'input, e 2^8 combinazioni possibili per l'output. Per ogni configurazione di input ho 2^8 output possibili:

$$\text{Numero funzioni} = (2^8)^{2^8}$$

Esercizio da esame.

1. Spiegare in cosa consiste il cifrario RSA e dimostrarne la correttezza
2. Darne un esempio di applicazione impiegando parametri numerici molto piccoli per cifrare il messaggio costituito dalle due cifre meno significative del proprio numero di matricola.

Esercizio da esame.

Posto che si scopra un algoritmo polinomiale per calcolare la funzione di Eulero, spiegare in termini matematici quale influenza la scoperta avrebbe sul cifrario RSA.

Risoluzione Nelle lezioni di teoria la prof.ssa ha insistito moltissimo sul fatto che fattorizzazione e risoluzione della funzione di Eulero siano problemi computazionalmente equivalenti. L'RSA si regge sul fatto che la fattorizzazione sia *One-Way Trapdoor*: risolvere la funzione di Eulero in tempo polinomiale sempre significa rompere il cifrario.

Esercizio da esame.

Si consideri un cifrario RSA con $p = 7, q = 11, e = 13$.

1. Determinare il valore della chiave privata d
2. Qual è la dimensione dei blocchi per la cifratura?
3. Cifrare 100011001010.

Risoluzione

- Per prima cosa calcoliamo

$$\phi(n) = (p - 1) \cdot (q - 1) = 6 \cdot 10 = 60$$

Successivamente individuiamo, attraverso l'algoritmo di Euclide esteso, l'inverso moltiplicativo di e modulo $\phi(n)$ (siamo certi che esiste - e che la soluzione sia unica - visto che $\text{MCD}(13, 60) = 1$)

$$d = e^{-1} \pmod{\phi(n)} = 13^{-1} \pmod{60}$$

Poniamo per comodità una delle formule tipiche di Euclide esteso

$$\langle d, x, y \rangle = \left\langle d', y', x' - \left\lfloor \frac{a}{b} \right\rfloor y' \right\rangle$$

Procediamo

$$\begin{aligned} \text{MCD}(13, 60) & < 1, 5, -23 > \longrightarrow < 1, -23, \dots > \\ \text{MCD}(60, 13) & < 1, -3, 5 > \longrightarrow < 1, 5, -3 - 4 \cdot 5 > = < 1, 5, -23 > \\ \text{MCD}(13, 8) & < 1, 2, -3 > \longrightarrow < 1, -3, 2 - 1 \cdot (-3) > = < 1, -3, 5 > \\ \text{MCD}(8, 5) & < 1, -1, 2 > \longrightarrow < 1, 2, -1 - 1 \cdot 2 > = < 1, 2, -3 > \\ \text{MCD}(5, 3) & < 1, 1, -1 > \longrightarrow < 1, -1, 1 - 1 \cdot (-1) > = < 1, -1, 2 > \\ \text{MCD}(3, 2) & < 1, 0, 1 > \longrightarrow < 1, 1, 0 - 1 \cdot 1 > = < 1, 1, -1 > \\ \text{MCD}(2, 1) & < 1, 1, 0 > \longrightarrow < 1, 0, 1 - 2 \cdot 0 > = < 1, 0, 1 > \\ \text{MCD}(1, 0) & < 1, 1, 0 > \end{aligned}$$

Abbiamo trovato l'inverso moltiplicativo

$$d = -23 \pmod{60} = 37$$

- Per la dimensione dei blocchi basta ricordare la seguente formula, vista nella teoria

$$\lfloor \log_2 n \rfloor = \lfloor \log_2(p \cdot q) \rfloor = \lfloor \log_2 77 \rfloor = 6$$

- Per prima cosa dividiamo la sequenza in blocchi di 6

$$m_1 = (100011)_2 = (35)_{10} \quad m_2 = (001010)_2 = (10)_{10}$$

Applichiamo la formula

$$m_1 = 35^{13} \pmod{77} \quad m_2 = 10^{13} \pmod{77}$$

In entrambi i casi applichiamo l'algoritmo delle esponenziazioni veloci. In entrambi casi scomponiamo 13 così:

$$13 = 1 + 8 + 4 = 2^0 + 2^3 + 2^2$$

inoltre $\lfloor \log_2 13 \rfloor = 3$.

(a) Per quanto riguarda m_1

$$\begin{aligned}35^1 \bmod 77 &= 35 \\35^2 \bmod 77 &= (35)^2 \bmod 77 = 70 \\35^4 \bmod 77 &= (70)^2 \bmod 77 = 49 \\35^8 \bmod 77 &= (49)^2 \bmod 77 = 14\end{aligned}$$

In conclusione

$$\begin{aligned}m_1 = 35^{13} \bmod 77 &= [(35 \bmod 77)(35^8 \bmod 77)(35^4 \bmod 77)] \bmod 77 = \\&= (35 \cdot 14 \cdot 49) \bmod 77 = (63)_{10} = (0111111)_2\end{aligned}$$

(b) Per quanto riguarda m_2

$$\begin{aligned}10^1 \bmod 77 &= 10 \\10^2 \bmod 77 &= (10)^2 \bmod 77 = 23 \\10^4 \bmod 77 &= (23)^2 \bmod 77 = 67 \\10^8 \bmod 77 &= (67)^2 \bmod 77 = 23\end{aligned}$$

In conclusione

$$\begin{aligned}m_2 = 10^{13} \bmod 77 &= [(19 \bmod 77)(10^8 \bmod 77)(10^4 \bmod 77)] \bmod 77 = \\&= (19 \cdot 23 \cdot 67) \bmod 77 = (10)_{10} = (1010)_2\end{aligned}$$

A questo punto concateniamo le sequenze ottenute per ottenere il crittogramma finale.

$$c = 0111111001010$$

Esercizio da esame.

Si consideri il cifrario RSA con chiave pubblica $n = 55, e = 7$.

1. Cifrare il messaggio $m = 10$.
2. Forzare il cifrario trovando p, q, d .
3. Decifrare il crittogramma $c = 35$.

Risoluzione

1. La consegna non ci fornisce p e q , ma direttamente n . Poiché abbiamo n ed e (che costituiscono la chiave pubblica) possiamo cifrare il messaggio con l'apposita

formula:

$$c = m^e \bmod n = 10^7 \bmod 55 = 10$$

2. La forzatura è possibile perchè sono stati scelti due valori p, q molto bassi. Osservando $n = 55$ possiamo dedurre al volo $p = 5, q = 11$, cioè

$$n = p \cdot q = 5 \cdot 11 = 55$$

A questo punto diventa facile calcolare la funzione di Eulero e quindi trovare d

$$\phi(n) = (p - 1)(q - 1) = 4 \cdot 10 = 40$$

$$d = e^{-1} \bmod \phi(n) = 7^{-1} \bmod 40 = 23$$

Come al solito facciamo il calcolo ricorrendo ad Euclide esteso.

3. Per decifrare il messaggio dobbiamo applicare la formula

$$m = c^d \bmod n = 35^{23} \bmod 55 =$$

Con l'algoritmo delle esponenziazioni veloci otteniamo

$$m = (35 \cdot 15 \cdot 5 \cdot 20) \bmod 55 = 30$$

Esercizio da esame.

Si consideri il protocollo basato sull'algoritmo DH con $g = 3$ e $p = 353$, e siano $x = 97$ e $y = 233$. Calcolare X, Y e la chiave k .

Risoluzione

- La consegna fornisce la coppia $\langle p, g \rangle$. Sappiamo che il protocollo richiede che i due interlocutori si mettano d'accordo su due valori: un numero intero primo p molto grande e un generatore g . Trovare i generatori non è cosa semplice: in molti casi si prendono le coppie suggerite dalle linee guida del NIST (non è un problema se si conosce la coppia adottata dai due interlocutori).
- X e Y sono i valori calcolati individualmente dai due interlocutori

$$X = g^x \bmod p = 3^{97} \bmod 353 = 40$$

$$Y = g^y \bmod p = 3^{233} \bmod 353 = 248$$

Si ricorre all'algoritmo delle esponenziazioni successive.

- La chiave si calcola nei seguenti modi (ciascun interlocutore lo fa per conto suo)

$$k = X^y \bmod p$$

$$k = Y^x \bmod p$$

cioè

$$k = g^{x \cdot y} \bmod p = 3^{97 \cdot 233} \bmod 353 = 160$$

Anche qua si ricorre all'algoritmo delle esponenziazioni successive.

Esercizio da esame.

Due utenti A, B vogliono costruire una chiave segreta di sessione impiegando il protocollo basato sull'algoritmo DH. A tale scopo concordano su una coppia pubblica di interi $\langle p, g \rangle$, con $p = 11$ (p è piccolo per costruire il nostro esempio), e $g = 6$.

1. Dimostrare che la coppia $\langle 11, 6 \rangle$ è adatta per il protocollo DH.
2. Posto che A e B scelgano come numeri “casuali” segreti x, y la terza e la quarta cifra del numero di matricola del candidato, creare la chiave di sessione indicando i calcoli eseguiti da A e B.

Risoluzione

- Consideriamo che un valore p piccolo vada bene ai fini della nostra simulazione (normalmente deve essere molto grande), l'importante è che sia primo (e 11 lo è). Per quanto riguarda g abbiamo detto che deve essere generatore dell'insieme \mathbb{Z}_p^* : la verifica mediante tabella è semplice vista la dimensione di p .

x	1	2	3	4	5	6	7	8	9	10
$6^x \bmod 11$	6	3	7	9	10	5	8	4	2	1

Abbiamo una permutazione dell'insieme \mathbb{Z}_p^* , inoltre si conferma la proprietà che $g^{\phi(n)} \bmod p = 1$

- Supponiamo che il nostro numero di matricola sia 654321. Le cifre centrali sono $x = 4$ e $y = 3$. Usiamole per trovare X ed Y

$$X = 6^4 \bmod 11 = 9 \quad Y = 6^3 \bmod 11 = 7$$

A questo punto si ottiene la chiave nei modi spiegati anche nell'esercizio precedente $k = 6^{4 \cdot 3} \bmod 11 = 3$

Esercizio da esame.

Il punto $P = (4, 7)$ appartiene alla curva ellittica $y^2 = x^3 - 5x + 5$ sui numeri reali?

Risoluzione Esercizio banale, ci basta sostituire x ed y , verificando che primo e secondo membro coincidano

$$7^2 = 4^3 - 5 \cdot 4 + 5 \implies 49 = 49$$

Coincidono, quindi il punto appartiene alla curva ellittica.

Esercizio da esame.

La curva ellittica di equazione $y^2 = x^3 + 10x + 5$ definisce un gruppo su \mathbb{Z}_{17} ? E sui reali?

Risoluzione Sia per quanto riguarda l'insieme dei numeri reali che per quanto riguarda \mathbb{Z}_{17} andiamo a verificare che la condizione di gruppo Abeliano sia soddisfatta.

- **Numeri reali.**

$$4a^3 + 27b^2 \neq 0 \implies 4(10)^3 + 27(5)^2 = 4675$$

Possiamo definire il gruppo Abeliano.

- **Insieme \mathbb{Z}_{17} .**

$$(4a^3 + 27b^2) \bmod 17 \neq 0 \implies (4(10)^3 + 27(5)^2) \bmod 17 = 0$$

La condizione non è soddisfatta.

Esercizio da esame.

Nella curva ellittica sui reali $y^2 = x^3 - 36x$, siano $P = (-3, 9)$ e $Q = (-2, 8)$. Trovare $P + Q$ e $2P$.

Risoluzione

- Per quanto riguarda $P + Q$ andiamo ad applicare le formule

$$\begin{cases} \lambda = \frac{y_Q - y_P}{x_Q - x_P} = \frac{8-9}{-2-(-3)} = \frac{-1}{1} = -1 \\ x_S = \lambda^2 - x_P - x_Q = 1 - (-3) - (-2) = 6 \\ y_S = \lambda(x_P - x_S) - y_P = -1(-3 - 6) - 9 = 0 \end{cases} \implies P + Q = (6, 0)$$

- Per quanto riguarda $2P$ consideriamo il caso $P = Q$. Applichiamo le formule, dove differisce solo λ

$$\begin{cases} \lambda = \frac{3x_P^2 + a}{2y_P} = \frac{3(-3)^2 - 36}{2 \cdot 9} = -\frac{1}{2} \\ x_S = \lambda^2 - x_P - x_Q = \left(-\frac{1}{2}\right)^2 - (-3) - (-3) = \frac{25}{4} \\ y_S = \lambda(x_P - x_S) - y_P = -\frac{1}{2}(-3 - \frac{25}{4}) - 9 = -\frac{35}{8} \end{cases} \implies 2P = \left(\frac{25}{4}, -\frac{35}{8}\right)$$

Esercizio da esame.

Calcolare gli opposti dei seguenti punti su curva ellittica su \mathbb{Z}_{17} :

$$P = (5, 8)$$

$$Q = (3, 0)$$

$$R = (0, 6)$$

Risoluzione Nella risoluzione, considerando la simmetria rispetto a $\lfloor \frac{p}{2} \rfloor$, agiamo esclusivamente su y nel modo seguente

$$-P = (5, -8 \bmod 17) = (5, 9) \quad -Q = (3, 0) \quad -R = (0, -6 \bmod 17) = (0, 11)$$

Esercizio da esame.

Nella curva ellittica $E_{17}(1, 7)$, siano $P = (1, 3)$ e $Q = (2, 0)$. Trovare $P + Q$ e $2P$.

Risoluzione Applichiamo le formule indicate dalla professoressa, ma attenzione al modulo! Soprattutto ricordarsi che quando parliamo di modulo parliamo di numeri interi e positivi (nel trovare $2P$ dobbiamo trovare per forza l'inverso moltiplicativo)!

- Per quanto riguarda $P + Q$

$$\begin{cases} \lambda = \frac{y_Q - y_P}{x_Q - x_P} \bmod 17 = \frac{0 - 3}{2 - 1} \bmod 17 = -3 \bmod 17 = 14 \\ x_S = (\lambda^2 - x_P - x_Q) \bmod 17 = ((14)^2 - 1 - 2) \bmod 17 = 6 \\ y_S = (\lambda(x_P - x_S) - y_P) \bmod 17 = (14(1 - 6) - 0) \bmod 17 = 12 \end{cases}$$

Otteniamo $P + Q = (6, 12)$

- Per quanto riguarda $2P$ consideriamo il caso $P = Q$

$$\begin{cases} \lambda = \frac{3x_P^2 + a}{2y_P} \bmod 17 = \frac{3 \cdot 1^2 + 1}{2 \cdot 3} \bmod 17 = \frac{4}{3} \bmod 17 = (4 \cdot 3^{-1}) \bmod 17 = 12 \\ x_S = (\lambda^2 - x_P - x_Q) \bmod 17 = (12^2 - 1 - 1) \bmod 17 = 6 \\ y_S = (\lambda(x_P - x_S) - y_P) \bmod 17 = 12(1 - 6) - 3 \bmod 17 = 5 \end{cases}$$

Nel calcolo di λ abbiamo calcolato l'inverso moltiplicativo $3^{-1} \bmod 17$, che è uguale a 3. Otteniamo $2P = (6, 5)$

Esercizio da esame.

Determinare i punti appartenenti alla curva ellittica $E_{11}(1, 6)$

Risoluzione Come abbiamo già detto non è possibile determinare il numero di punti con una formula precisa. Determinare proprio i punti che costituiscono la curva ellittica si ottiene come nell'esempio visto in teoria: individuazione dei residui quadratici e prova di tutti i valori di x possibili. Ricordiamo che la curva è rappresentata nel seguente modo (forma normale di Weierstrass)

$$y^2 = (x^3 + x + 6) \bmod 11$$

- **Calcolo residui quadratici.**

y	0	1	2	3	4	5	6	7	8	9	10
$y^2 \bmod 11$	0	1	4	9	5	3	3	5	9	4	1

Otteniamo che i residui quadratici sono: 0, 1, 3, 4, 5, 9

- **Test dei possibili valori di x .**

Testiamo i possibili valori di x , se troviamo corrispondenza con uno dei residui quadratici recuperiamo i valori di y che hanno generato quel residuo quadratico.

$x = 0$	$y^2 = 6$	Non ho residuo quadratico, no soluzione
$x = 1$	$y^2 = 8$	Non ho residuo quadratico, no soluzione
$x = 2$	$y^2 = 5$	$y = 4, 7 \rightarrow (2, 4), (2, 7)$
$x = 3$	$y^2 = 3$	$y = 5, 6 \rightarrow (3, 5), (3, 6)$
$x = 4$	$y^2 = 8$	Non ho residuo quadratico, no soluzione
$x = 5$	$y^2 = 4$	$y = 2, 9 \rightarrow (5, 2), (5, 9)$
$x = 6$	$y^2 = 8$	Non ho residuo quadratico, no soluzione
$x = 7$	$y^2 = 4$	$y = 2, 9 \rightarrow (7, 2), (7, 9)$
$x = 8$	$y^2 = 9$	$y = 3, 8 \rightarrow (8, 3), (8, 8)$
$x = 9$	$y^2 = 7$	Non ho residuo quadratico, no soluzione
$x = 10$	$y^2 = 4$	$y = 2, 9 \rightarrow (10, 2), (10, 9)$

Conclusioni: l'ordine della curva è 13 e l'insieme dei punti che costituiscono la curva ellittica è il seguente

$$\begin{aligned} E_{11}(1, 6) = & \{(2, 4)(2, 7), (3, 5), (3, 6), (5, 2), (5, 9), (7, 2) \\ & (7, 9), (8, 3), (8, 8), (10, 2), (10, 9)\} \cup \{0\} \end{aligned}$$

Esercizio da esame.

Nella curva ellittica $E_{23}(14, 12)$, sia $P = (1, 2)$. Calcolare $11P$

Risoluzione

1. Scomponiamo la costante $k = 11$ in potenze di 2

$$11P = (2^0 + 2^1 + 2^3)P = (1 + 2 + 8)P = P + 2P + 8P$$

2. Calcoliamo i raddoppi. Sappiamo che dovremo calcolare, oltre a P un numero di elementi pari a $\lfloor \log_2 11 \rfloor = 3$.

$$2P = (6, 17)$$

$$4P = 2(2P) = 2(6, 17) = (0, 14)$$

$$8P = 2(4P) = 2(0, 14) = (6, 6)$$

3. Prendiamo solo i raddoppi che ci interessano e sommiamo

$$11P = P + 2P + 8P = (1, 2) + (6, 17) + (6, 6) = (2, 18) + (6, 6) = (1, 2)$$

I calcoli sono stati ottenuti con le formule indicate dalla professoressa. L'ordine in cui sommiamo gli elementi non è un problema, visto che abbiamo definito un Gruppo Abeliano (dove è garantita in primis la proprietà commutativa).

Enigma della discordia.

Decifrare

LOCE LOCE LOCE

LOCE LOCE LOCE LOCE LOCE LOCE LOCE LOCE LOCE

Suggerimento: 5,6

Risoluzione Problema da settimana enigmistica, giochino fatto dalla Bernasconi con gli studenti di un anno accademico passato. Si osservi che:

- Prima riga LOCE è scritto tre volte. Seconda riga LOCE è scritto nove volte. Si ottiene TRE e NOVE
- TRE e NOVE, assieme a LOCE, restituiscono la stringa

TRENOVELOCE \Rightarrow TREN VELOCE

- La parola TREN è costituita da cinque caratteri, la parola VELOCE da sei (riferimento al suggerimento).