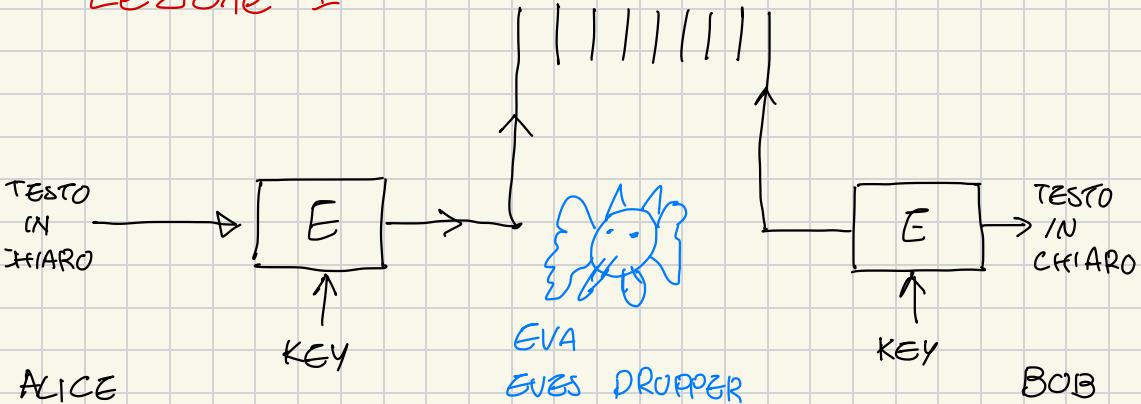




Lezione 1



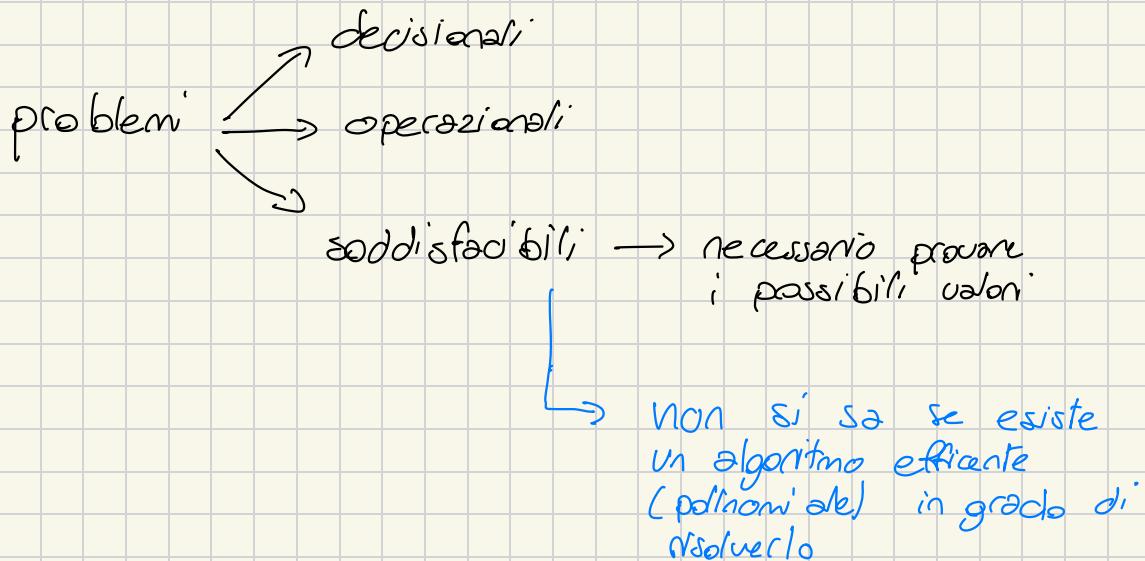
NECESSARIO COSTRUIRE UN PROTOCOLLO INATTACCABILE
MA COSA VOLERE È ATTACCABILE?

CRITTOGRAFIA ELLITTICA

- ↳ più sicura
- ↳ più veloce
- ↳ back door → via d'ingresso alternativa sconosciuta agli altri
- ↳ usa una matematica diversa
- ↳ basata sulle curve ellittiche

A parità di lunghezza le chiavi della crittografia ellittica sono più sicure.
DH $\{n\} \rightarrow$ numero

Ha bisogno di meno risorse



Se esiste una soluzione efficiente al problema decisionale ne esiste una anche alla soluzione del problema operazionale

La crittografia ci dice:

Questo sistema è sicuro nella base in cui non esiste una soluzione semplice a quest'altro problema ↳ similare all'idea di riduzione funzionale (FONDAMENTI)

GENERAZIONE NUMERI PSEUDOCASUALI

$$x_{i+1} = ax_i \bmod b$$

dove a e b sono numeri relativamente primi tra loro

I numeri così generati sono crittograficamente non sicuri
Un numero è crittograficamente sicuro se si può riconoscere alla sua catena solo sapendo il seed

Un cifrario perfetto si ottiene quando l'entropia della chiave K è maggiore o uguale alla chiave del testo

STEGANOGRAFIA → nasconde il messaggio → non so dove sia nascosto il messaggio
ma viene trasmesso in chiaro

- OGGI NON SONO VENUTO A FABULARE PERCHE' E' MEGLIO DA-
- RE UNA BUONA LEZIONE DI CORREZIONE

CRITTOANALISI → MALE

↳ si occupa di ricavare informazioni dai ciphertext senza conoscere la chiave

CRITTOGRAFIA → BENE

↳ vuole nascondere le informazioni

Quando un sistema è sicuro?

Tutto può essere infinto

Un sistema è sicuro quando il costo necessario ad ottenere l'informazione è maggiore del valore dell'informazione stessa

Lezione 2

to hack → smarritare



hacker → buoni, attaccano i sistemi e poi forniscono informazioni con cui sistemare

cracker → cattivi, vogliono attaccare i system

script kiddies → attaccanti che non sanno quello che fanno, lanciano di tutto per cercare di creare danni

Creiamo un codice → costo creazione



costo per attaccarlo

Non ci si basa sulla segretezza degli algoritmi per ottenere la segretezza, ma ci basiamo sulla segretezza della chiave

Noi sappiamo che una chiave è sicura in base a quanti non riescano ad attaccarlo

MAI INCURRE ALTRE PERSONE A VOLERCI ATTACCARO

CIFRARIO DI CESARE

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

La chiave è lo sfasamento

Alfabeto scuotuto in modo che dopo la Z ci sia la A

B U O N G I O R N O
E X R Q J L R U Q R

Algoritmo $C = D(k, p) = (p+k) \bmod 26$

Con il cifrario di cesare abbiamo solo 26 chiavi possibili;
con un attacco a forza bruta si riesce facilmente ad infangare
il crittosistema

Possiamo utilizzare analisi delle frequenze dei caratteri del ciphertext confrontando i risultati con l'analisi delle frequenze delle lingue per risalire più facilmente al plaintext

CRITTOSSISTEMA:

- ① PLAINTEXT: testo in chiaro
- ② ALGORITMO DI CIFRATURA: trasforma il testo in chiaro in testo cifrato
- ③ CHIAVE SEGRETA: è un input dell'algoritmo di cifratura, ha valore indipendente
- ④ CIPHER TEXT: testo prodotto dall'algoritmo di cifratura dipende dal testo in chiaro e dalla chiave
- ⑤ Decryption algorithm: trasforma il testo cifrato in testo in chiaro

Requisiti per l'uso della crittografia convenzionale

- 1 Necessità di un algoritmo di cifratura forte. Ciò significa che chi possiede testi cifrati non sia in grado di trovarne facilmente la chiave per poterli decifrare.
- 2 Il mittente e il destinatario devono aver ricevuto copie della chiave segreta mediante un canale sicuro. Se qualcuno trovasse la chiave conoscendo l'algoritmo, l'intera comunicazione diventerebbe leggibile.

Nella cifratura simmetrica, l'algoritmo di cifratura non deve rimanere segreto, ma solo la chiave deve esserlo. (Caratteristica pratica per un uso diffuso)

Elementi essenziali di una cifratura simmetrica

- sorgente che produce un messaggio di testo in chiaro $X = [x_1, x_2, \dots, x_n]$
- viene generata una chiave di cifratura $K = [k_1, k_2, \dots, k_s]$, che deve essere mantenuta segreta
- con il messaggio X e la chiave K come input, l'algoritmo di cifratura genera il testo cifrato $Y = [y_1, y_2, \dots, y_n]$ rappresentato come $y = E(K, x)$
- il destinatario inteso, in possesso della chiave, può decifrare il messaggio: $X = D(K, y)$

I sistemi crittografici possono essere caratterizzati lungo tre dimensioni indipendenti:

- 1 Il tipo di operazioni utilizzate per trasformare il testo in chiaro in testo cifrato. Il testo in chiaro viene trasformato in testo cifrato utilizzando sostituzione e trasposizione degli elementi che lo compongono (bit, lettere...), assicurando che tutte le operazioni siano reversibili. Molti sistemi implicano fasi multiple di queste operazioni.
- 2 Il numero di chiavi utilizzate. Abbiamo due tipi principali di sistemi di cifratura: uno simmetrico, con una chiave condivisa e l'altro asimmetrico con chiavi diverse per mittente e destinatario (sistema a due chiavi o cifratura a chiavi pubblica).
- 3 Il modo in cui il testo viene processato. Le cifrature possono essere a blocchi o a flusso. Nella cifratura a blocchi, il testo in chiaro viene elaborato a blocchi, generando un blocco di testo cifrato per ciascun blocco in input. Nella cifratura a flusso, l'elaborazione avviene in modo continuo, producendo l'output elemento per elemento.

CHIAVE CORPO DI LETTURA (Cifrario di Playfair)

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

BULONIGI ORINO

cx NA IK NM AN

chiavi 25!

c'è una scorciatoia

Assumiamo pacman-effect, ovvero come se a destra di R ci fosse M ecc.

Potremmo fare un analisi sulle frequenze delle coppie di lettere

PERMUTAZIONI ARBITRARIE

Π funzione di permutazione dell'alfabeto

$$E(x) = \Pi(x)$$

$$D(c) = \Pi^{-1}(c)$$

26! chiavi

Per migliorare la tecnica monoalfabetica è possibile utilizzare diverse sostituzioni monoalfabetiche man mano che si procede attraverso il messaggio in chiaro. Il nome generico per questo approccio è cifrario di sostituzione polialfabetico. Tutte queste tecniche hanno le seguenti caratteristiche in comune:

1. Un insieme di regole di sostituzione monoalfabetica corrente viene utilizzato
2. Una chiave determina quale particolare regola viene scelta per una data trasformazione

c'è scorciatoia, possiamo nuovamente fare analisi delle frequenze
c'è sempre corrispondenza lettera ugual - lettera ugual

Cifrario di Vigenère (PASWD)

Cifrario polialfabetico

P A S W D P A S W D P A S W D

P R O V A D I C O D I F I C A

codifica le lettere con il
cifrario di Cesare con chiave
corrispondente alla lettera sopra

E R G R D S I U R G X F A Y D

Ho più chiavi:

Non è più possibile fare analisi frequenze, non c'è più corrispondenza
che a lettera uguale corrisponde lettera uguale

Se so la lunghezza della password però posso fare comunque
analisi delle frequenze perché a distanze uguali, lettera uguale
corrisponde a lettera uguale

P A S W D P A S W D P A S W D P A S W D
↑↑↑↑↑ ↑↑↑↑↑ ↑↑↑↑↑ ↑↑↑↑↑

se nelle posizioni dello
stesso indice ho le
stesse lettere, pure il
ciphertext sarà uguale

↳ posso fare analisi
delle frequenze

Se la password è lunga tanto quanto il testo ho un **cifrario perfetto** e non posso più fare analisi delle frequenze

Cifrario di Vernam

CESARE SU ALFABETO BINARIO

ONE TIME PAD

			Key				
	0	1	0	0	1	1	
K=0	0	1	P	0	1	0	1
			plaintext				

↳ Cifrario monouso è un sistema crittografico che utilizza una chiave casuale di lunghezza uguale o maggiore del messaggio da crittografare.
È un sistema di crittografia perfetto, nel senso che il messaggio cifrato non può essere decifrato o violato senza conoscere la chiave.

	0	1	C	0	1	1	0
K=1	1	0					

↓
or
esclusivo

(abbiamo applicato lo XOR)

L'algoritmo di decodifica è lo stesso di quello di codifica
Semplificazione del cifrario di Cesare su un alfabeto binario

vettore
chiave

\bar{K} 0 1 1 0 1 0

\bar{P} 1 1 0 1 1 0

vettore
plaintext

\bar{C} 1 0 1 1 0 0

vettore
ciphertext

È uno schema di Vigenere dove la lunghezza della chiave è pari a
a quella del testo

Stesso principio solo con due simboli

Ma siamo sicuri che guardando N ciphertext non so nulla?

$$C = X \oplus K$$

↘ cipher text ↗ plain text ↓ XOR ↗ chiave
 ↗

K composto da bit scelti a caso, lungo quanto il messaggio. Monouso
 x scelto a caso dall'utente

Sia P prob $[x = z]$

probabilità che
key sia 0

$$P_C [C=0] = P_x [x=0] \cdot P_k [k=0] + P_x [x=z] \cdot P_k [k=z]$$

probabilità che
cipher text
sia 0

probabilità che
plain text sia 0

probabilità
che plain text
sia 1

$$= (1-p) \cdot \frac{1}{2} + p \cdot \frac{1}{2} = \frac{1}{2}(1-p+p) = \frac{1}{2}$$

A prescindere da ciò che è il plaintext il ciphertext è puramente casuale.
 Perciò non posso ottenere informazioni sul ciphertext.

ONE TIME PAD è uno schema perfetto, dal ciphertext non possiamo dire nulla sul plaintext senza avere la chiave. Sicurezza statistica

La chiave però è lunghissima ed è monouso
 ↳ costo eccessivo

Inoltre la chiave con probabilità $\frac{1}{2^n}$
 ↳ lunghezza
della stringa

ENTROPIA

L'entropia misura il grado d'incertezza medio dei messaggi emessi da una sorgente

Quindi rappresenta una stima del numero medio di bit necessari a codificare un messaggio proveniente da quella sorgente

Hartley

Entropia della variabile X su un insieme discreto e con eventi equiprobabili

$$H(x) = \log_2 N$$

$H(x)$ indica il numero minimo di bit necessari a codificare il messaggio in maniera sicura

Shanon

Definiamo Entropia della variabile X su un insieme discreto e con eventi non equiprobabili

$$H(x) = \sum_{x \in N} -P_i \log(P_i)$$

↙
numero minimo di
bit necessari a
codificare il messaggio
in maniera sicura

↙
valore negativo
/
valore positivo

Hartley è solo un caso particolare di Shanon

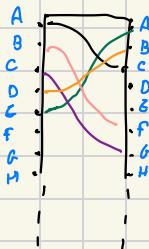
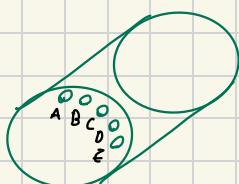
Lezione 3

permutazione

$$\Pi : A \rightarrow A$$

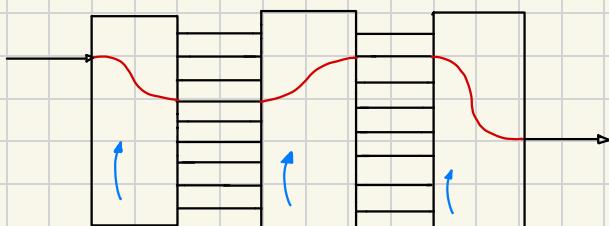
$$\Pi_1 \circ \Pi_2 \circ \Pi_3$$

Applicazione composta di permutazioni
corrisponde ad una sola permutazione



○ → sono i contatti

MACCHINA ENIGMA (tentata di decodificare da Turing)



Una macchina a rotori è una macchina crittografica che sfrutta la crittografia a sostituzione polialfabetica

La macchina è composta da cilindri rotanti; ognuno con 26 pin di input e 26 pin d'output

3 rotori, prendo il primo carattere e lo codiflico, poi ruoto il rotore di una posizione e codiflico il prossimo, così per 26 volte, poi ruoto il penultimo rotore e così via.

Known Ciphertext Attack (KCA)

Tale sistema protegge dall'analisi delle frequenze poiché per 26^3 permutazioni non è possibile fare un'analisi delle frequenze

Conoscendo il testo cifrato dobbiamo risalire al testo in chiaro

Known Plaintext Attack (KPA)

Conoscendo sia il testo in chiaro, sia il testo cifrato dobbiamo risalire alla chiave

Se dico che un sistema è resistente al KPA dico che è più sicuro rispetto a un sistema resistente al KCA, più un sistema resiste a un attacco potente più è sicuro

Chosen Plaintext Attack

Il testo viene deciso dall'attaccante e cifrato, si deve scoprire la chiave

Adaptive Chosen Plaintext

Ho un approccio adattivo quindi in base al ciphertext ottenuti posso produrre nuovi plaintext per formulare le prossime cifrature

Enigma era vulnerabile al Known Plaintext attack, infatti si è scoperto che i messaggi iniziavano tutti con la stessa formula

DATA ENCRYPTION STANDARD (DES)

Sistema con chiavi da 56 bit, al tempo molto sicuro rispetto ai sistemi utilizzati

Tempo di vita stimato 20 anni

Sostituito successivamente dall'Advanced Encryption Standard (AES)

Vantaggi:

- era implementabile su un chip

- algoritmo estremamente rapido

DES è stato attaccato una ventina di anni dopo la sua implementazione

Lunghezza della chiave: 56 bit. Abbiamo 2^{56} possibili chiavi differenti.

Lunghezza del blocco: opera su blocchi a 64 bit.

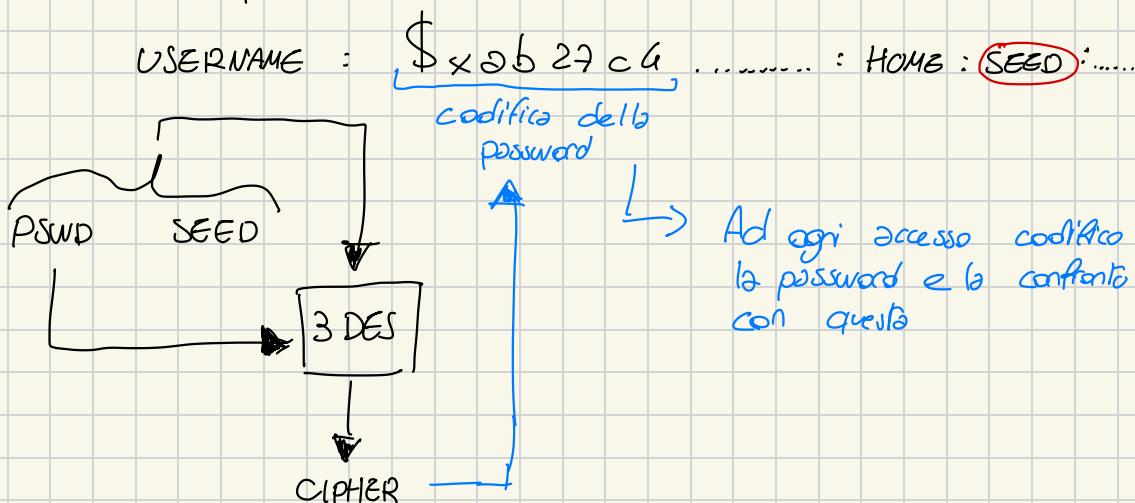
Attacco con tecnica di CRITTOANALISI DIFFERENZIALE

Mandavano due messaggi e calcolavano la differenza tra i bit del ciphertext.

Creavano equazioni lineariamente indipendenti sui bit della chiave

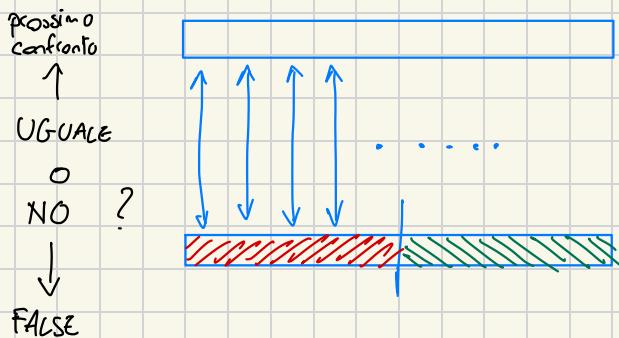
44 equazioni differenziali sui bit della chiave,
 $(56 - 44) = 12$ bit scoperti, trovano la chiave
in 2^{12} tentativi
 \downarrow
2048

Codifica password in Mux



IBM 360, mainframe super utilizzato in passato

Algoritmo di confronto fra stringhe



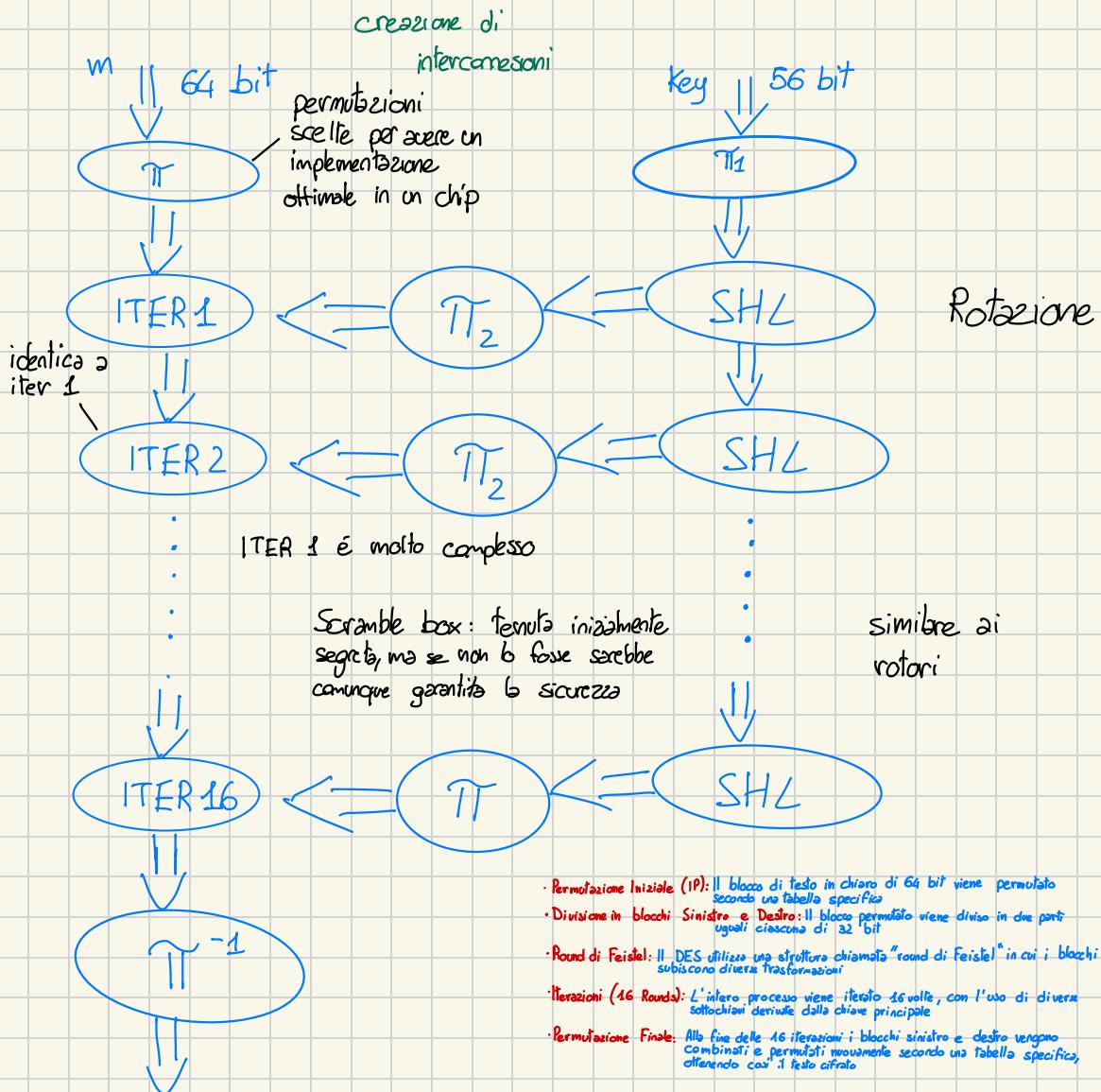
password inserite dall'utente vengono divise e scritte su due pagine diverse, di cui una fuori dalla cache

- se ho cache miss la parte di password nella pagina della cache è corretto
- se non ho cache miss la parte di password nella pagina della cache è errata

Alternative, posso non guardare i cache hit e i cache miss e guardare: analisi del tempo, di consumo potenza

Cifrario a blocchi

Funzionamento 3 DES



- **Permutazione Iniziale (IP):** Il blocco di testo in chiaro di 64 bit viene permuto secondo una tabella specifica.
- **Divisione in blocchi Sinistro e Destro:** Il blocco permuto viene diviso in due parti uguali ciascuna di 32 bit.
- **Round di Feistel:** Il DES utilizza una struttura chiamata "round di Feistel" in cui i blocchi subiscono diverse trasformazioni.
- **Iterazioni (16-Rounds):** L'intero processo viene iterato 16 volte, con l'uso di diverse sottotabelline derivate dalla chiave principale.
- **Permutazione Finale:** Alla fine delle 16 iterazioni i blocchi sinistro e destro vengono combinati e permessi nuovamente secondo una tabella specifica, offrendo così il testo cifrato.

Ciphertext

Algoritmo di codifica identico
a quello di decodifica
Idea di Feistel

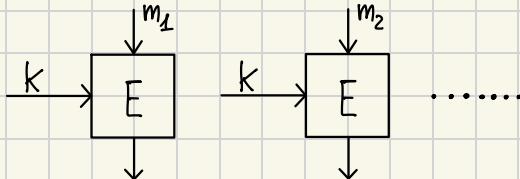
Come posso cifrare stream di blocchi ?

Un cifrario a flusso crittografa un flusso di dati digitali: un bit o un byte alla volta.

Electronic Code book

ECB

prendo il messaggio M e lo divido nei blocchi che lo compongono



Malleabilità → proprietà pericolosa, posso modificare il testo sapendo il plaintext

Esempio

Pattern di testo in chiaro simili generano pattern di testo cifrato simili

→ TRASF | 1000 E | A ROB |

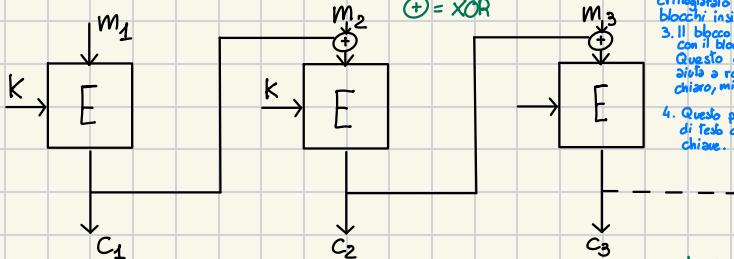
→ TRASF | 1 M E | A PIPPO |

• Posso fare scambi di blocchi tra i messaggi in maniera nota

• Se ci sono messaggi ripetuti me ne accorgo

• Messaggi che iniziano uguali me ne accorgo

Cipher Block Chaining



introduce un certo grado di indipendenza tra i blocchi di testo in chiaro durante il processo di crittografia. Funziona così seguendo:

1. Prima di crittografare viene generato un vettore di inizializzazione casuale (iv). Questo vettore è combinato con il primo blocco di testo in chiaro tramite un'operazione di XOR.

2. Il risultato di questa operazione XOR viene quindi crittografato utilizzando l'algoritmo di crittatura a blocchi insieme alla chiave.

3. Il blocco di testo cifrato risultante viene poi combinato con il blocco di testo successivo prima della crittografia. Questo collegamento tra i blocchi di testo in chiaro aiuta a rompere la correlazione tra i blocchi di testo in chiaro, migliorando la sicurezza rispetto a ECB.

4. Questo processo continua garantendo che ciascun blocco di testo cifrato dipenda dal blocco precedente e dalla chiave.

La decodifica avviene seguendo lo stesso processo in ordine inverso.

Posso ancora scambiare i blocchi ma non so il risultato

Come decodifico?

È un metodo simmetrico di cifratura quindi la procedura di decodifica è la medesima di quella di codifica

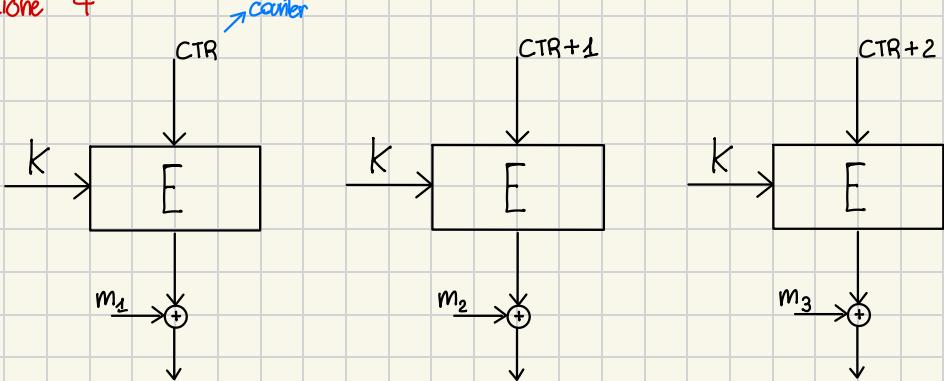
Come vengono gestiti i messaggi ripetuti?

Mi accorgo di messaggi ripetuti
Inserisco un blocco random a inizio messaggio
per ogni messaggio ci sono $2^{\text{lung. blocco di numeri}}$ possibili doppiioni

faccio XOR con valori casuali

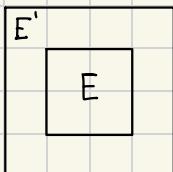
Tuttavia non riusciamo a spedire stream, vorrei trovare sistemi per spedire stream

Lezione 4



possiamo spedire i bit senza dover aspettare

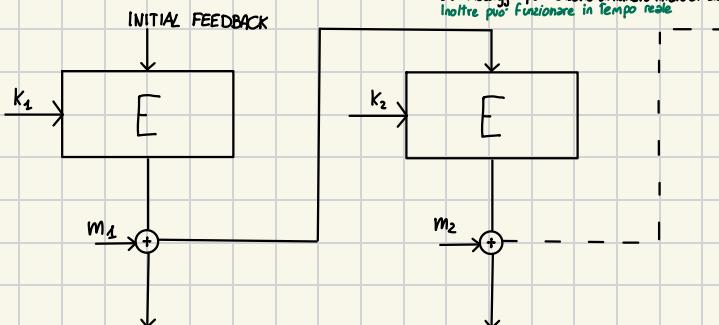
dico accordarmi sul valore di CTR



Se E non è staccabile
Compango E in E' , se E' è staccabile
allora lo era anche E

Ci rifaremo spesso a questo schema

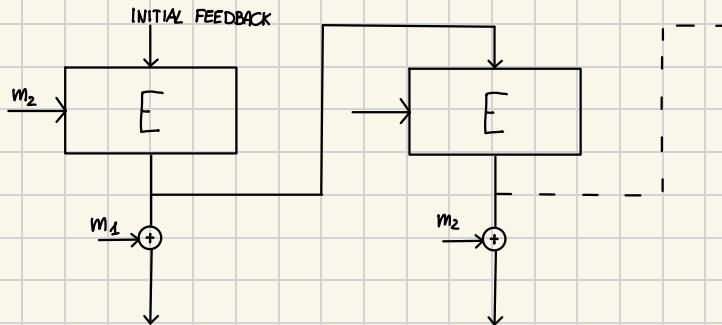
CIPHER FEEDBACK \longrightarrow cifrario a flusso: elimina la necessità di aggiungere padding
a un messaggio per renderlo un numero intero di blocchi
Inoltre può funzionare in tempo reale



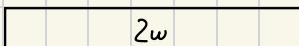
Assomiglia molto a cipherblock chaining tuttavia l'output del primo blocco viene messo in input al prossimo passaggio

Lavorando su un messaggio noto (E') prima dello XOR lavoro su un messaggio noto quindi posso spedire un qualsiasi numero di bit

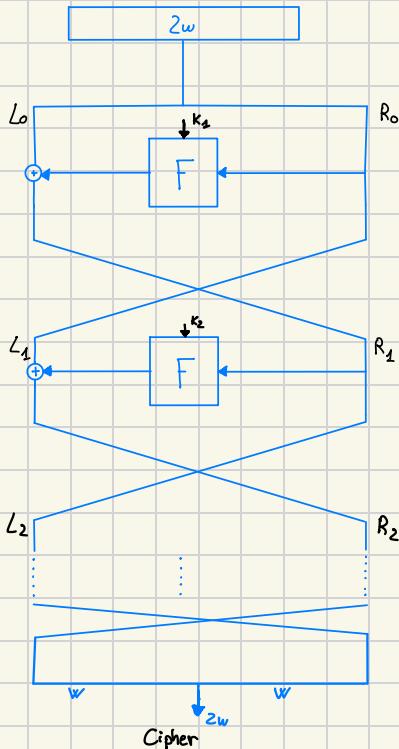
Output Feedback



FAISTEL → tecnica di cifratura a blocchi



L'input $2w$ è un numero pari di bit → viene diviso in due metà



A destra applico il blocco di

Feistel → funzione di trasformazione che dipende da una sottochiave specifica del round

Faccio XOR di L_0 con il risultato del blocco di Feistel che dà in input w bit

progettato per ridurre una complessità tale da rendere il processo di crittanalisi complesso e costoso

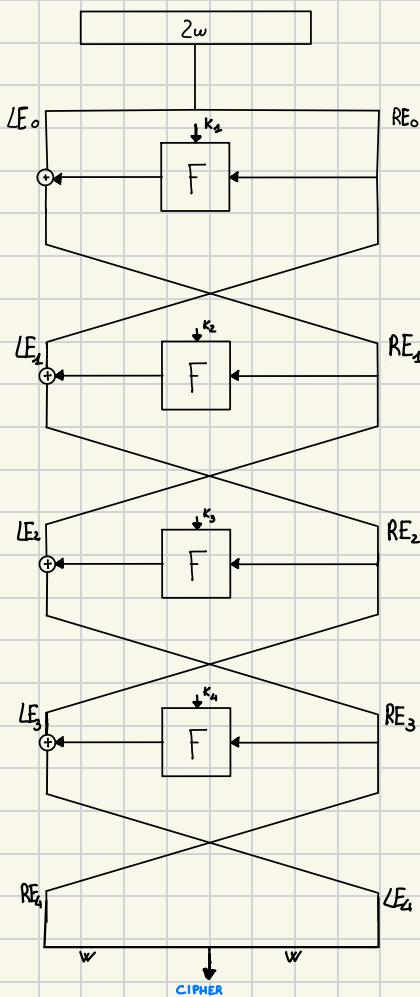
Le chiavi e le scatole di Feistel nel caso di DES sono i valori che restano, però in realtà lo schema di Feistel funziona qualsiasi sia la scatola F

Se cifro un plaintext con Feistel ottengo ciphertext

Se uso questo ciphertext con Feistel e utilizzo le chiavi al contrario ottengo il plaintext

Dobbiamo dimostrarlo

C'è un ultimo scambio prima di dare la risposta dopo un numero prefissato di round



Così facendo ottengo sempre i risultati dello schema precedente

$$LD_1 = RD_0 = LE_4 = RE_3$$

$$RD_1 = LD_0 \oplus F(RD_0, K_4)$$

$$= RE_4 \oplus F(RE_3, K_4)$$

$$= LE_3 \oplus F(RE_3, K_4) \oplus F(R_3, K_4)$$

SER
tra due logici
di: zero

Affinché questo schema funzioni devo dunque "solo" scrivere bene lo scatola

Così facendo ottengo sempre i risultati dello schema precedente

Diffusione

Struttura statistica del plaintext viene dissipata

$$m_1, m_2, m_3, \dots, m_n$$

↓
lettera

$y_n \leftarrow \sum_{i=1}^n m_i \quad \text{MOD } 25$

considero le lettere come numeri

Le operazioni all'interno dell'algoritmo di Feistel assicurano che anche piccoli cambiamenti nel testo in chiaro si propagino in modo significativo attraverso i vari round, garantendo che piccole modifiche nel testo in chiaro producano cambiamenti significativi nel testo cifrato.

y_1, y_2, \dots, y_n ottenuto un testo distribuito in maniera uniforme

Necessario che F sia robusta e ben progettata

AES costruito con moduli in grado di eliminare ciascuno un tipico attacco

Confusione

Obiettivo ogni bit di input deve influenzare tutti i bit di output

Quando cambio un bit mi aspetto mediamente almeno la metà combi

Il problema è scambiare la chiave

chiave pubblica - chiave privata

Crittografia moderna → si cerca di partire da problemi difficili su cui hanno lavorato in passato
 ↓
 si parla della teoria dei numeri

La funzione F introduce confusione nel testo in chiaro, in modo che la relazione fra il testo cifrato e la chiave sia complessa e non lineare. Questo rende difficile per un crittoanalista estrarre informazioni significative sulla chiave o sul testo in chiaro, anche se conosce la relazione fra il testo cifrato e la chiave.

Reversibilità: l'architettura di Feistel consente di effettuare facilmente l'operazione inversa (decifratura) con gli stessi componenti dell'algoritmo di cifratura. Questa proprietà di "reversibilità" semplifica notevolmente il processo di decifratura senza compromettere la sicurezza del sistema.

TEORIA DEI NUMERI → studio degli interi e delle loro relazioni

$$(A, *)$$

$$*: A \times A \rightarrow A$$

Complessità computazionale: La scelta di una F complessa e sufficientemente caotica rende la crittoanalisi computazionalmente costosa e difficile, richiedendo risorse computazionali significative, per eseguire con successo un attacco di crittoanalisi.

Associativa $\forall a, b, c \in A \quad (a * b) * c = a * (b * c)$

Neutro $\exists e \in A \quad \forall a \in A \quad a * e = e * a = a$

Inverso $\forall a \in A \quad \exists a^{-1} \quad a * a^{-1} = a^{-1} * a = e$

$$\cancel{\forall x = a^{-1} b}$$

$$\cancel{\forall x = A^{-1} b}$$

Ogni volta che ci viene data un'operazione matematica che ha queste proprietà si parla di **GRUPPO**

per numeri naturali:
 Un'operazione si dice chiusa se l'operazione applicata a due numeri naturali restituisce un numero naturale.

$$\mathbb{N} \circ \mathbb{N} \rightarrow \mathbb{N}$$

funzione one way \rightarrow facile da calcolare
 \hookrightarrow difficile da invertire

Classe di equivalenza

$$\mathbb{Z}_n \equiv_n$$

insieme dei numeri moduli

prendiamo le classi di equivalenza di questa operazione e abbiamo \mathbb{Z}_n

$a \equiv_n b$ SSE a e b divisi per n danno lo stesso resto

insiemi chiusi

$$a \equiv b \pmod{n} \iff (a \bmod n) = (b \bmod n)$$

esempio

$$5 = 11 \pmod{3} \text{ perché entrambi hanno resto 2 nella divisione per 3.}$$

Noi informatici dobbiamo lavorare con gruppi finiti perché lavoriamo con i bit

Quando abbiamo una relazione di equivalenza possiamo costituire l'insieme degli oggetti equivalenti tra loro. L'insieme degli insiemi di oggetti equivalenti forma una partizione, gli elementi di tale partizione sono detti classi di equivalenza e si denonno mediante parentesi quadre di un elemento di tale classe. Ogni singola classe di equivalenza può essere rappresentata da qualsiasi elemento della classe stessa.

$$\frac{1}{2} \equiv \frac{3}{6}$$

$$\mathbb{Z}_n = \{0, 1, 2, 3, \dots, n-1\}$$

$$10.00 \pmod{12} \equiv 22.00$$

Lezione 5

$$a/n = b \text{ con resto } c$$

↓
a mod n risulta c

$$a_2 \equiv a_2 \text{ mod } n$$

↓ congruo

Se e solo se il resto

$$a = bn + c$$

a è nella classe di c, vuol dire che a è rappresentabile come un multiplo di n più resto c

$$a \equiv c \text{ mod } n$$

↓
il resto della divisione n
è uguale al ↓ resto della divisione per n

Se ho una relazione di equivalenza posso costruire una classe di equivalenza

$$[a]_n \text{ una classe di equivalenza è data dagli insiemi equivalenti ad a}$$

$$[0], [1], [2], \dots, [n-1]$$

classe di equivalenza modulo n

$$a = kn + \underline{\underline{b}}$$

posso trovare un k per cui _____
b è uno di quei valori

Algebra modulare

$\mathbb{Q} \rightarrow$ razionali

$$(a, b) \approx (c, d)$$

SSE

$$ad = bc$$

$\mathbb{Z}_{\text{mod } n}$

In generale la somma di due classi di equivalenza è la classe di equivalenza della somma dei rappresentanti

$$[a] + [b] \stackrel{\Delta}{=} [a+b]$$

class 5 class 4 avendo \mathbb{Z}_7 $9 \text{ mod } 7 = 2$

5+4 = 9 = 2

→ somma è classe 9

0 è l'elemento neutro

Proprietà valide per l'operazione di somma sull'insieme \mathbb{Z}_n

• Commutativa, Associativa, Elemento neutro, Elemento inverso

L'operazione \mathbb{Z}_n con l'operazione di somma forma un Gruppo Abeliano

$$[a] + [n-a] = [0]$$

In generale la moltiplicazione di due classi di equivalenza è la classe di equivalenza del prodotto dei rappresentanti

$$(Z_n, +)$$

$$[a] * [b] \triangleq [a * b]$$

$$|Z_n| = \#(Z_n) = n$$

$$(Z_n, *)$$

NEUTRO $[1]$

$$Z_6 = \{0, 1, 2, 3, 4, 5\}$$

Gli interi con la moltiplicazione non sono un gruppo poiché non c'è l'inverso

$1X$

$$(Z_n^*, *)$$

$$L = \{a \in Z_n \mid \text{MCD}(a, n) = 1\}$$

sono coprimi
non hanno fattori
in comune se non 1

$$Z_{15}^* = \{1, 2, 4, 7, 8, 11, 13, 14\}$$

abbiamo gli inversi perciò è un GRUPPO

$$xa + yb = \text{MCD}(a, b)$$

2)

$$xa + yn = 1$$

Teorema di Eulero

Per ogni $a, b \exists x, y \quad ax + by = \text{mod}(a, b)$

Se $a \in Z_n^*$ allora $\text{mcd}(a, n) = 1$ per definizione e quindi

$$ax + ny = 1$$

$$ax = 1 - ny$$

$$ax \equiv 1 \pmod{n}$$

Quindi la classe d'equivalenza di x è l'inverso moltiplicativo di a . La necessità di lavorare con gruppi nasce dal fatto che in informatica è necessario lavorare con insiemi finiti, in questo caso algebre su insiemi finiti, in particolare sui gruppi, in modo da manipolare gli elementi in base alle proprietà

Proprietà valide per l'operazione di moltiplicazione sull'insieme Z_n

Comutativo, Associativo, Elemento neutro

L'operazione Z_n con l'operazione di moltiplicazione forma un semigruppo

$b < a$
$\text{MCD } a \text{ e } b$
$\text{MCD } (a-b), a$
$ab \quad \underline{a-b} \quad a$

$$xa = 1 - yn$$

$$\mathbb{Z}_p^* = \{1, 2, 3, \dots, p-1\}$$

perché tutti i numeri più piccoli di p
sono coprimi con p essendo p primo

$$|\mathbb{Z}_p^*| = p-1$$

$$\mathbb{Z}_p^* = \{1, 2, 3, \dots, p-1\}$$

Sia G un gruppo (\mathbb{Z}_p^*) e sia g un elemento di G

$$\begin{array}{ccccccccc} \mathbb{Z}_n^* & 2 & 4 & 8 & 5 & 10 & 9 & 7 & 3 & 6 & 1 \\ g = 2 & 2 & 4 & 8 & 5 & 10 & 9 & 7 & 3 & 6 & 1 \\ g^2 & g^4 & g^8 & g^5 & g^{10} & g^9 & g^7 & g^3 & g^6 & g^1 & g^0 \end{array}$$

leggermente diverso
è l'esponente da dare
alla base per ottenere
un numero

g=2 in
questo esempio

2 è un generatore di G

Non tutti gli elementi di \mathbb{Z}_p^* generano \mathbb{Z}_p^* basta guardare cosa accade con

$$3, 9, 5, 1$$

sottogruppo del gruppo originale l'elemento considerato non è un generatore

Cardinalità del gruppo

$$\varphi(n) \cong |\mathbb{Z}_n^*|$$

$$\varphi(n) = \begin{cases} n-1 & n \text{ primo} \\ (p-1)(q-1) & n = pq \end{cases}$$

$$[\alpha(n, g, a)]_{\varphi(n)}$$

$$a = g^i = g^{i + k\varphi(n)} \quad \forall k$$



Quando un elemento a è quadrato?

$$\exists x. \quad a = x \cdot x$$

In \mathbb{Z}_p^* metà degli elementi è un quadrato

TUTTI GLI ELEMENTI g^{2i} SONO QUADRATI

$$\begin{aligned} g^i &\xrightarrow{-1} g^{i + \frac{p-1}{2}} \\ (g^i)^{-1} \cdot g^{i + \frac{p-1}{2}} &= g^{2i} \end{aligned}$$

cardinalità
del gruppo

Generatore primi

Se prendo \mathbb{Z}_n^* con n primo, allora \mathbb{Z}_n^* è ciclico, poiché ogni elemento di \mathbb{Z}_n^* è un generatore di \mathbb{Z}_n^* .

In generale un numero primo non può essere scampato in fattori, quindi:

$$\mathbb{Z}_p^* = \{1, 2, 3, \dots, p-1\}$$

Esempio

• 1

$$2 \cdot 2 = 4 \equiv 1 \pmod{7}$$

$$3 \cdot 3 = 9 \equiv 2, 2 \cdot 3 = 6, 6 \cdot 3 = 18 \equiv 4, 4 \cdot 3 = 12 \equiv 5, 5 \cdot 3 = 15 \equiv 1 \pmod{7}$$

$$4 \cdot 4 = 16 \equiv 2, 2 \cdot 4 = 8 \equiv 1$$

$$5 \cdot 5 = 25 \equiv 4, 4 \cdot 5 = 20 \equiv 6, 6 \cdot 5 = 30 \equiv 2, 2 \cdot 5 = 10 \equiv 3, 3 \cdot 5 = 15 \equiv 1$$

$$6 \cdot 6 = 36 \equiv 1 \pmod{7}$$

Abbiamo che \mathbb{Z}_7^* è ciclico, poiché esiste un generatore che genera tutto il gruppo, in questo caso 3 e 5.

Densità dei numeri primi

La densità dei numeri primi è inversamente proporzionale al numero di bit che compongono il numero.

Supponiamo di avere un numero casuale n di k bit, allora la probabilità che n sia primo è $\frac{1}{k}$.

Supponiamo di voler comporre un numero n di k bit, utilizzando un algoritmo che mi genera tale numero.

Per verificare che tale numero sia primo, utilizzo un algoritmo casuale che sceglie casualmente un numero a tra 1 e $n-1$ e verifica che il numero scelto sia primo. Statisticamente circa la metà dei numeri scelti sono testimoni del fatto che n non sia primo. Se il test fallisce e dice che il numero n non è primo, allora termino. Se il test ha esito positivo allora scelgo un altro numero a e ripeto il test di primalità. Se tutte le volte che salgo un numero a il test ha esito positivo, allora la probabilità di accettare la primalità di n è $\frac{1}{2^k}$, dove k è il numero di test.

Logaritmo Discreto

Supponiamo di avere a disposizione un gruppo \mathbb{Z}_p^* , un generatore g e un elemento di tale gruppo $a \in \mathbb{Z}_p^*$, visto che le potenze del generatore enumeralo l'intero gruppo, ci sarà un potenza x che mi permette di ottenere a :

$$g^x = a = g^{x+k(p-1)}$$

L'oggetto x è detto logaritmo discreto di a in base g . Trouare il logaritmo discreto di un numero è un problema difficile, non possiamo dire che non esistano algoritmi efficienti per calcolarlo, ma non ne conosciamo nessuno.

Visto che non ne conosciamo nessuno possiamo utilizzare il logaritmo come funzione one-way. Difficoltà aumenta con dimensione del gruppo.

Numeri Quadrati

Se ho un gruppo G con un'operazione binaria \otimes , un elemento $a \in G$ è detto quadrato se e solo se esiste un $x \in G$ tale che $x \otimes x = a$.

Quindi esiste una radice quadrata di a in G . Gli elementi in \mathbb{Z}_p^* sono $p-1$, ma solo la metà di questi sono quadrati. $\frac{p-1}{2}$. Perché un numero sia un quadrato devo trovare un numero del gruppo che elevato al quadrato mi dà il numero dell'insieme; il numero risultante sarà un quadrato.

Esempio

$$\mathbb{Z}_7^* = \{1, 2, 3, 4, 5, 6\}$$

• $1 \cdot 1 = 1$ quindi 1 è un quadrato

• $2 \cdot 2 = 4$ quindi 4 è un quadrato

• $3 \cdot 3 = 9 \equiv 2$ quindi 2 è un quadrato

• $4 \cdot 4 = 16 \equiv 2$ quindi 2 è un quadrato

• $5 \cdot 5 = 25 \equiv 4$ quindi 4 è un quadrato

• $6 \cdot 6 = 36 \equiv 1$ quindi 1 è un quadrato

Gli elementi che non sono quadrati sono $\{3, 5, 6\}$, quindi esattamente la metà degli elementi del gruppo sono quadrati.

Teorema

Tutti gli elementi g^i sono quadrati in \mathbb{Z}_p^*

Se consideriamo un generatore g di \mathbb{Z}_p^* e generiamo l'intero gruppo, è chiaro che tutti gli elementi che sono potenze pari di g sono quadrati. Di conseguenza tutti gli elementi che sono potenze dispari di g non sono quadrati.

L'elemento g^{2i} ha due radici quadrate, g^{2i} e $g^{2i+\frac{p-2}{2}}$

$$g^{2i} = \begin{cases} g^{2i} \\ g^{2i+\frac{p-2}{2}} \end{cases}$$

$$(g^{2i+\frac{p-2}{2}}) = g^{2i} \cdot g^{p-i} = g^{2i} \cdot 1 = g^{2i}$$

So che le radici quadrate sono anche

$$g^i = \begin{cases} g^i \\ g^{i+\frac{p-2}{2}} \end{cases}$$

Quindi elevare un generatore alla cardinalità del gruppo mi dà 1, ma elevarlo alla metà della cardinalità del gruppo mi dà -1

Con questa osservazione possiamo costruire un algoritmo che mi permetta di distinguere gli elementi che sono quadrati da quelli che non lo sono. Se conosciamo il logaritmo discreto di un numero a in base g , il fatto che non siamo in grado di calcolare il logaritmo discreto in maniera efficiente non preclude la possibilità che esistano altri algoritmi.

SIMBOLO DI LEGENDRE

Il simbolo di Legendre

$$\left(\frac{a}{p}\right) \triangleq a^{\frac{p-1}{2}} \pmod{p}$$

Allora essendo g^{2i} un quadrato

$$\left(g^{2i}\right)^{\frac{p-1}{2}} = g^{\frac{2i(p-1)}{2}} = (g^i)^{p-1} \stackrel{\text{cardinalità del gruppo}}{=} 1^{p-1} = 1 \pmod{p}$$

Sia $a = g^{2i+1}$ ovvero un non quadrato, allora

$$\left(g^{2i+1}\right)^{\frac{p-1}{2}} = \left(g^{2i}\right)^{\frac{p-1}{2}} \cdot g^{\frac{p-1}{2}} = (g^i)^{p-1} \cdot g^{\frac{p-1}{2}} = 1 \cdot (-1) = -1 \pmod{p}$$

Quindi applicando il simbolo di Legendre ad un numero a in base g otteniamo

$$\left(\frac{a}{p}\right) = \begin{cases} 1 & \text{se } a \text{ è un quadrato} \\ -1 & \text{se } a \text{ non è un quadrato} \end{cases}$$

Quindi il simbolo di Legendre mi permette di distinguere gli elementi che sono quadrati da quelli che non lo sono.

Sappiamo che nessuno conosce algoritmi efficienti per calcolare il logaritmo discreto, se però dire se un numero è un quadrato o meno osservando l'ultimo bit del numero.

Con il simbolo di Legendre posso calcolare il bit meno significativo di un numero in maniera efficiente, nonostante non siamo capaci di calcolare il logaritmo discreto.

Il fatto che non siamo capaci di invertire una funzione non implica che non siamo capaci di calcolare qualche bit della funzione.

Il nostro obiettivo però è quello di non ricavare informazioni da nulla, neanche un bit. Quindi: l'elevamento a potenza in \mathbb{Z}_p^* non potrà essere usato per codificare

LEZIONE 6

\mathbb{Z}_n^*

$$n = p \cdot q$$

Come si calcola l'esponentiale di un numero modulo n?

X

$$b = 1 \ 0 \ 1 \ 1 \ 0 \ 1$$

$$b = 2^5 + 2^3 + 2^2 + 2^0$$

Iterative Squaring

Con il simbolo di Legendre possiamo calcolare in maniera polinomiale, ma non in maniera efficiente.

Se eleviamo un numero a^b per b non operiamo con un algoritmo efficiente.

In \mathbb{Z}_p^* è possibile calcolare in tempo polinomiale un'esponentiale a^b con $a, b \in \mathbb{Z}_p^*$

Rappresentiamo b in base 2:

$$b = \sum_{i=0}^k b_i \cdot 2^i$$

Quindi:

$$a^b = \sum_{i=0}^k b_i \cdot a^{2^i} = \prod_{i=0}^k a^{2^i b_i}$$

$$x^2 \quad x^2 \quad x^2 \quad x^2 \quad x^2 \quad x \quad x$$

Calcoliamo iterativamente i quadrati ottenuti e moltiplichiamo quelli che servono

Faccendo 5 moltiplicazioni ho trovato x elevato al peso del vari bit

$$\begin{array}{c} (5) \\ x^2 \\ x^2 \end{array} \quad \begin{array}{c} (3) \\ x^2 \\ x^2 \end{array} \quad x^2 \quad x$$

Faccio $2K$ operazioni al posto di 2^K

L'algoritmo per moltiplicare due numeri a K bit la complessità è $O(K^2)$
-> numero grande

SHIFT A DESTRA IN BINARIO \rightarrow DIVISO PER 2

Con il sistema di numerazione binario possiamo calcolare in maniera efficiente l'esponenziazione, in un numero di moltiplicazioni pari al numero di bit di b

Procedure Iterative Squaring (a,b)

Exp (x, b) → tiene i quadrati iterativi

y ← 1

t ← x

while b ≠ 0

if ($b \% 2^1 = 0$) then y ← y * t

b ← b / 2

t ← t * t

ret y

L'algoritmo ha un problema
il numero di bit di y
aumenta in maniera indeterminata
per formare le potenze
inserite in

Exp (x, b, n)

y ← 1

t ← x

while b ≠ 0

if ($b \% 2^1 = 0$) y ← y * t mod n

b ← b / 2

t ← t * t mod n

ret y

A questo punto non ho più il problema di avere numeri con crescita esponenziale
del risultato, i risultati parziali rimangono sempre nella stessa quantità di bit

Il gruppo \mathbb{Z}_n^* con $n = p \cdot q$

TEOREMA

Un numero x è quadrato in \mathbb{Z}_n^* se e solo se è quadrato in \mathbb{Z}_p^* e in \mathbb{Z}_q^*

$$|\mathbb{Z}_n^*| = |\mathbb{Z}_p^*| \cdot |\mathbb{Z}_q^*| = (p-1) \cdot (q-1)$$

Calcolando quindi gli elementi coprimi con n

Ho un algoritmo per verificare che x sia quadrato in \mathbb{Z}_n^*

Esiste una funzione $\varphi(n)$ che mi fornisce la cardinalità di \mathbb{Z}_n^* , chiamata funzione di Eulero $\varphi(n) \cong |\mathbb{Z}_n^*|$

Ma se ho n e non p e q?

Nessuno capisce algoritmi che riconoscono quadrati di numeri che sono moltiplicazioni di numeri primi

In \mathbb{Z}_n^* ci sono $\frac{\varphi(n)}{4}$ elementi che sono quadrati

$$\frac{\varphi(n)}{4}$$

$\frac{\varphi(n)}{4}$ sono quadrati in \mathbb{Z}_p^* e non in \mathbb{Z}_q^*

$$\frac{\varphi(n)}{4}$$

$\frac{\varphi(n)}{4}$ sono quadrati in \mathbb{Z}_q^* e non in \mathbb{Z}_p^*

$$\frac{\varphi(n)}{4}$$

$\frac{\varphi(n)}{4}$ non lo sono in nessuno

Teorema

Ogni quadrato ha 4 radici $\pm x$ e $\mp y$

Potrei dimostrare un algoritmo polinomiale in \mathbb{Z}_n^* in grado di calcolare le radici quadrate

TUTTO SI BASA SUL FATTO CHE FATTORIZZARE n in p e q è difficile

Quindi per capire se un numero è un quadrato in \mathbb{Z}_n^* in maniera semplice è necessario conoscere la fattorizzazione di n. Bisogna calcolare il simbolo di Legendre rispetto a p e q e so il risultato.

Se non conosco la fattorizzazione di n non conosco algoritmi efficienti per stabilire se un numero è un quadrato in \mathbb{Z}_n^* non disponendo della fattorizzazione di n. Quindi anche la quadraticità di un numero è un problema difficile

Alessio dimostriamo (probabile scrittura) che se riuscissimo a calcolare le radici quadrate allora riusciremmo a fattorizzare n

Siano $\pm x$ e $\pm y$ le radici di a
di C

$$\text{ALLORA} \quad \text{MCD}(x+y, n) = p \circ q$$

Dimostrazione

$$x^2 \equiv y^2$$

$$x^2 - y^2 \equiv 0$$

$$(x+y)(x-y) \equiv 0$$

$$(x+y)(x-y) = kn$$

$$\text{sia } p \mid (x+y)$$

\downarrow
questo
divide \uparrow

è possibile che $q \mid (x+y)$?

Supponiamo di avere un algoritmo che mi dà una radice di n

ALGORITMO FATTORIZZA (n)

$$a \in_{\mathbb{R}} \mathbb{Z}_n^*$$

prendi uniformemente un elemento
di \mathbb{Z}_n^*

$$\frac{(p-1)(q-1)}{pq} > \frac{1}{2} \cdot \frac{1}{2}$$

Sia x la radice quadrata di a^2

$$x \leftarrow \sqrt{a^2}$$

if $x \neq \pm a$ rct $\text{mcd}(a+x, n)$

questo algoritmo fattorizza
 n con probabilità $\frac{1}{2}$

Se io voglio generare un numero primo genero un numero casuale di K bit, ho un algoritmo probabilistico casuale che mi dice se è primo, se non è primo lo genero nuovamente e procedo

Densità dei numeri primi si ottiene sia simile a $\log n = K$

Consequently $\frac{1}{K}$ volte trovi il numero primo

Costruire un numero primo ha complessità in media sul K^4

Come capiamo se un numero è quadrato o meno in

SIMBOLO DI JACOBI → quando è 1 è difficile sapere la quadraticità

è una generalizzazione del simbolo di Legendre

0 vale 1
-1 vale -1

$$\left(\frac{\alpha}{n}\right)$$

$$\left(\frac{\alpha}{n_1 n_2}\right) = \left(\frac{\alpha}{n_1}\right) \left(\frac{\alpha}{n_2}\right)$$

$$\left(\frac{\alpha}{p}\right) \left(\frac{\alpha}{q}\right)$$

$$\left(\frac{\alpha_1 \alpha_2}{n}\right) = \left(\frac{\alpha_1}{n_1}\right) \cdot \left(\frac{\alpha_2}{n_2}\right)$$

$$\mathbb{Z}_n^*$$

Ad oggi stabilire se un numero è un quadrato in \mathbb{Z}_n^* quando il simbolo di Jacobi è -1 allora il numero non è quadrato in \mathbb{Z}_n^* per definizione

Il simbolo di Jacobi di un numero è calcolabile in tempo polinomiale

Sapendo che il simbolo di Jacobi si calcola in tempo polinomiale cosa si può dire sulla quadraticità di quel numero?

Il simbolo di Jacobi ci dà qualcosa sulla quadraticità di A

Ad oggi quando il simbolo di Jacobi è 1 nessuno riesce a dire qualcosa sulla quadraticità

metà degli elementi di \mathbb{Z}_n^* ha probabilità 1/2

Trucco \rightarrow fai le cose a caso \rightarrow se ok va bene



se no ok rifai

La quadraticita' di un numero con simboli di Jacobi e' un altro dei problemi difficili.

$x \cdot y$

sia x quadrato e $\rightarrow x \cdot y$ quadrato
sia y quadrato

$\left(\frac{xy}{p} \right)$ un numero per essere quadrato deve essere
in $\mathbb{Z}_p^* < \mathbb{Z}_q^*$

$$\Rightarrow \left(\frac{x}{p} \right) \left(\frac{y}{p} \right) = 1 \cdot 1 = 1$$

Identico con \mathbb{Z}_q^*

Un numero e' quadrato in \mathbb{Z}_q^* se e solo se e' un quadrato in \mathbb{Z}_q^* e \mathbb{Z}_p^*

y non quadrato con $\left(\frac{y}{p} \right) = 1$ xy non quadrato

$$\left(\frac{xy}{p} \right) = \left(\frac{x}{p} \right) \left(\frac{y}{p} \right) = 1 \cdot (-1) = -1 \quad \left(\frac{xy}{q} \right) = \left(\frac{x}{q} \right) \left(\frac{y}{q} \right) = -1$$

Se x e y non quadrati con entrambi simboli di giacobi 1

Il simbolo di giacobi del prodotto è 1

$$\left(\frac{xy}{p} \right) = \left(\frac{x}{p} \right) \left(\frac{y}{p} \right) = (-1)(-1) = 1$$

xy è un quadrato su \mathbb{Z}_p^*

Idem su \mathbb{Z}_q^*

Quindi su \mathbb{Z}_n^* è quadrato

posso moltiplicare per non quadrati per invertire la quadraticità

posso moltiplicare per quadrati per mantenere la quadraticità

Moltiplicare un oggetto (quadrato o non quadrato) per un quadrato a caso mi permette di ottenere un valore casuale con la stessa quadraticità

Costruire non quadrato con simbolo di JACOBI 1 non è semplice

L'elevamento a 2) quadrato in \mathbb{Z}_n c'è one way TRAPDOOR

Facile da calcolare
difficile da invertire

se sappiamo
fattorizzare
però c'è
semplice

Turing riduzione

Lemma

Siano x e y due radici quadrate
di uno stesso quadrato di \mathbb{Z}_n^* , tali che
 $x \not\equiv \pm y$.

Allora il mcd tra $x+y$ e n è un
fattore di n

Dimostrazione

Sia $n = pq$, visto che x e y sono radici quadrate di uno stesso numero
Supponiamo che $x \equiv y \pmod{n}$. Quindi:

$$\begin{aligned} x \equiv y \pmod{n} &\Rightarrow x^2 - y^2 \equiv 0 \pmod{n} \\ &\Rightarrow (x-y)(x+y) \equiv 0 \pmod{n} \quad \text{per qualche } k \end{aligned}$$

Supponiamo che p divide $x+y$ ma è possibile che q divide $x-y$? Se fosse possibile allora n divide $x+y$, ma ciò vorrebbe dire che $x \equiv y \pmod{n}$ ovvero $x \equiv -y \pmod{n}$, che è assurdo perché $x \not\equiv \pm y$. Quindi $\text{mcd}(x+y, n) = p$.

Supponiamo che q divide $x+y$ ma è possibile che p divide $x-y$? Se fosse possibile allora n divide $x+y$, ma ciò vorrebbe dire che $x \equiv y \pmod{n}$, ovvero $x \equiv -y \pmod{n}$ e ciò è assurdo perché $x \not\equiv \pm y$. Quindi $\text{mcd}(x+y, n) = q$.

Supponiamo che p non divide $x+y$ e che q non divide $x+y$, ma p e q sono fattori del prodotto $(x-y)(x+y)$, quindi p e q devono essere fattori di almeno uno dei due fattori. Se p non divide $x+y$ allora q deve dividere $x+y$, allora p e q sono fattori di $x-y$, ma allora $x \equiv y \pmod{n}$ e ciò è assurdo.

Capitolo 2. Teoria dei numeri

40

Supponiamo che p divide $x-y$, ma è possibile che q dividere $x+y$? Se fosse possibile allora n divide $x+y$, ma ciò vorrebbe dire che $x \equiv y \pmod{n}$ ovvero $x \equiv -y \pmod{n}$, che è assurdo perché $x \not\equiv \pm y$. Quindi $\text{mcd}(x-y, n) = p$.

Supponiamo che q divide $x-y$, ma è possibile che p dividere $x+y$? Se fosse possibile allora n divide $x+y$, ma ciò vorrebbe dire che $x \equiv y \pmod{n}$, ovvero $x \equiv -y \pmod{n}$ e ciò è assurdo perché $x \not\equiv \pm y$. Quindi $\text{mcd}(x-y, n) = q$.

Supponiamo che p non divide $x-y$ e q non divide $x-y$, ma p e q sono fattori del prodotto $(x-y)(x+y)$, quindi p e q devono essere fattori almeno uno dei due fattori. Se p non divide $x+y$ allora q deve dividere $x-y$, allora p e q sono fattori di $x-y$, ma allora $x \equiv y \pmod{n}$ e ciò è assurdo.

□

Se abbiamo due radici quadrate distinte allora riusciamo a trovare la fattorizzazione di n . Il problema ora è come faccio a trovare due radici distinte di un numero se mi viene fornito l'algoritmo per il calcolo della radice quadrata? Supponiamo che esista un algoritmo $A \in \text{PPT}$ (*probabilistic polynomial time*) che calcoli la radice quadrata di un numero in \mathbb{Z}_n^* . Allora:

```
1: procedure FACTORIAL(n)
2:    $x \in_R \mathbb{Z}_n^*$ 
3:    $y \leftarrow A(x^2)$ 
4:    $z \leftarrow \text{mcd}(x-y, n)$ 
5:   if  $z \neq n$  then return  $z$ 
6:   else
7:     return Factorial( $n/z$ )
8: end if
8: end procedure
```

Prendo un quadrato a caso, di questo quadrato conosco una radice scelta uniformemente tra le quattro possibili. L'algoritmo A mi restituisce una radice quadrata di x^2 , l'algoritmo sceglierà la radice quadrata in qualche modo, sicuramente indipendente dalla scelta fatta su x . La probabilità che la radice scelta sia x o l'opposto sia $\frac{1}{2}$, quindi il test che verifica se z è un fattore di n avrà successo con probabilità $\frac{1}{2}$, quindi ripeto l'algoritmo A un numero costante di volte poiché la probabilità di successo è costante. Quindi l'algoritmo è polinomiale.

Il numero di esperimenti da eseguire per aver successo è data dalla distribuzione geometrica, e il valore atteso è il reciproco della ragione di successo, quindi in media devo eseguire due volte l'algoritmo A per avere successo.

Ciò ci porta a dire che calcolare la radice quadrata è verosimilmente difficile, perché se qualcuno ci riuscisse allora potremmo fattorizzare in tempo polinomiale.

L'idea di dimostrare la sicurezza di un crittosistema è quella di dimostrare che esista un algoritmo che utilizzi come sottoprocedura un algoritmo che risolve un problema difficile. I problemi difficili sono quelli che non si riescono a risolvere in tempo polinomiale, quindi il calcolo della radice quadrata, il logaritmo discreto e la fattorizzazione di numeri primi.

Lezione 7

Algoritmo di Diffie-Hellman (crittografia a chiave pubblica-chiave privata)

\mathbb{Z}_p^* , g → decisi dal National Institute for Security and Technology
 gruppo primo
 2ta
 p star
 generatore del gruppo
 elementi di \mathbb{Z}_p^*

A sceglie $x \in E_R$ $\{1, \dots, p-1\} \rightarrow$ chiave privata
 authority
 calcola $g^x \mod p \rightarrow$ chiave pubblica

L'idea è che la chiave pubblica sia nota a tutti mentre la privata è nota solo al proprietario

Codifica avviene con chiave pubblica
 decodifica con chiave privata

↳ Chiunque può cifrare un messaggio ma solo il destinatario può decifrarlo

Dalla chiave pubblica non possiamo calcolare la chiave privata perché è difficile calcolarne il logaritmo discreto

B sceglie $y \in E_R \{1, \dots, p-1\}$
 calcola $g^y \mod p$

A e B possono calcolare $g^{xy} \mod p$ e $g^{yx} \mod p$, che coincidono

Il sistema è sicuro perché calcolare $g^{xy} \mod p$ è complessionalmente inaffidabile. Avendo a disposizione g^x e g^y non è possibile calcolare g^{xy} . Se sappiamo rispondere al problema del logaritmo discreto, allora possiamo risolvere il problema di Diffie-Hellman.

In un sistema a chiave pubblica abbiamo i seguenti algoritmi:
 - Algoritmo di generazione delle chiavi: $G: \mathbb{Z}_p^k \rightarrow (P_k, S_k)$
 - Un algoritmo di encipher: $E: m, P_k \rightarrow E(m, P_k)$
 - Un algoritmo di decipher: $D: C, S_k \rightarrow D(C, S_k)$

Dove k è il
 security parameter
 che indica la
 lunghezza
 della chiave

Ovviamente vale la seguente relazione

$$\forall m \quad D(E(m, P_k), S_k) = m$$

$(g^y)^x$
 g^{xy} → Il numero risultante è il medesimo
 $(g^x)^y$ → Lo questa è la chiave utilizzata da A e B per comunicare

Decifrare un messaggio a partire dal testo cifrato dovrebbe essere difficile perché la chiave privata è complessionalmente inaffidabile

→ Hanno chiave comune vero come possiamo essere sicuri che nessun altro possa saperlo?

↳ Se supponiamo la presenza di un algoritmo probabilistico di calcolo del logaritmo discreto di un numero, è ovvio che sia un problema semplice perché basta calcolare il logaritmo discreto delle chiavi pubbliche

Per dimostrare che non siamo in grado di ottenere tale chiave bisogna fare una Turing-risaluzione e dimostrare che il problema di riferimento è legato al calcolo del logaritmo discreto

$$g^x g^y \mapsto g^{xy}$$

Non esiste dimostrazione che Diffie-Hellman sia effettivamente sicuro ma dopo 30 anni nessuno ha ancora attaccato, solo che ora furono chiavi molto lunghe, perciò spesso ora si usano gruppi su curve ellittiche

L'ipotesi di Diffie-Hellman, se uno ci pone una soluzione al problema non siamo in grado di dire o meno se essa sia una soluzione

Prendiamo un agente che lancia una moneta

$$\epsilon \quad b = \{0, 1\}$$

$$b \in_R \{0, 1\}$$

$$x, y, z \in_R \{1, \dots, p-1\}$$

$$b \rightarrow \begin{cases} (g^x, g^y, g^{xz}) & \text{se } b=0 \\ (g^x, g^y, g^z) & \text{se } b=1 \end{cases}$$

difficili da distinguere

Il concetto di "distinguibilità" è un concetto probabilistico, ovvero che la possibilità di poter distinguere due insiemi di elementi è trascurabile

In sostanza un attaccante con qualche informazione deve avere un vantaggio. E che DEVE essere trascurabile rispetto a un attaccante senza alcuna informazione

B

$$b' \leftarrow B(m, n, \ell)$$

$$P_{\epsilon} [B(m, n, \ell) = b] = \frac{1}{2} \rightarrow \begin{array}{l} \text{probabilità di} \\ \text{individuare il bit } b \end{array}$$

Ho un bit
a caso
sparsato

chimpo abbia una
potenza di calcolo
ordinaria non
riesce a notare la
differenza tra le
due triple

Riusciamo a indovinare con probabilità maggiore di $\frac{1}{2}$

$\frac{1}{(p-1)^3}$ l'algoritmo indovina e $1 - \frac{1}{(p-1)^3}$ indovina con probabilità $\frac{1}{2}$

$$\frac{1}{(p-1)^3} + \left(1 - \frac{1}{(p-1)^3}\right) \frac{1}{2}$$

$$\frac{1}{2} + \frac{1}{2(p-1)^3}$$

$$\frac{1}{2} + \frac{1}{2^{3k+1}}$$

sparando x, y, z a caso sto sparando
 k bit a caso per ciascuno dei tre
se indovino m, n, l calcolo le che
triplet di b e vedo quali delle due
ho indovinato altrimenti se $x, y, z \neq m, n, l$
indovino con probabilità $\frac{1}{2}$

Devo capire quanto è grande la probabilità che si
aggiunge a $\frac{1}{2}$ per capire quanto so

Sia $P \Pr[B(m, n, l) = b]$

Definitivamente non
penziona \Rightarrow ovvero non
importa all'inizio ma da un
certo punto in poi

\Rightarrow Costante

$|P - \frac{1}{2}| < K$

deve essere più
piccolo di qualsiasi
polinomio altrimenti
sono in grado di
costruire un algoritmo
polinomiale che indovina
con probabilità 1

vedo quanto
 p è distante
da $\frac{1}{2}$

numero di bit

\downarrow voglio che sia
piccolo quindi devo
dare poco vantaggio

Fissa un algoritmo e
fissa un polinomio, non
si riesce a indovinare
la soluzione a un problema

Se ho una dizione piccola nesso a indovinare la soluzione, perciò considero le dizioni più lunghe
di K

Rivest Shamir Adleman - RSA

g^x è una funzione one-way. Il protocollo di Diffie-Hellman
è sicuro se esiste una funzione one-way.

Le migliori sono trapdoor

p, q primi a caso con K bit

$$n = pq$$

$$e \in A \subset \mathbb{Z}_{\varphi(n)}^*$$

$$d = e^{-1} \pmod{\varphi(n)}$$

Scegliamo un elemento d co-primo con $\varphi(n)$

$$e: \text{MCD } (e, \varphi(n)) = 1$$

L'è numero coprimo

Encryption algorithm

$$E: m, (n, e) \mapsto m^e \pmod{n}$$

Description Algorithm

$$D: c, (n, d) \mapsto c^d \pmod{n}$$

Funzionamento

$$\begin{aligned} & m^e \\ & \text{ciphertext (CHIAVE PUB (n,d))} \\ & \text{decodifica (CHIAVE PRIV (n,d))} \\ & = (m^e)^d \pmod{\varphi(n)} \\ & = m^{ed} \pmod{\varphi(n)} \\ & = m^{ed} \pmod{n} \\ & = m^{ed} \pmod{n} \\ & = m \end{aligned}$$

L'unico modo per calcolare la radice e-esima
è calcolare d e decifrare il messaggio, quindi d
è l'informazione trapdoor

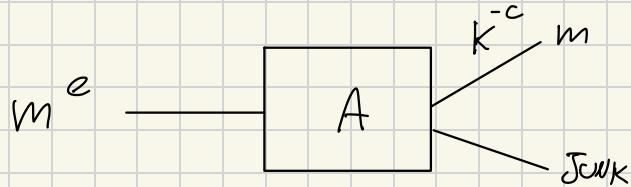
Si usa il fatto che
gli esponenti in un
gruppo sono ciclici

Infatti $d \cdot e$ è congruo a 1 modulo $\varphi(n)$, quindi $d \cdot e = k\varphi(n) + 1$
Per il teorema del resto chino, $m^{k\varphi(n)+1} \equiv m$, e non solo per
gli elementi di \mathbb{Z}^*

La funzione one-way è la funzione di codifica, quindi m^e ,
l'inverso di m^e è la radice e-esima ovvero $m = c^d$, ma per calcolare la
radice e-esima ad oggi non esiste un algoritmo efficiente

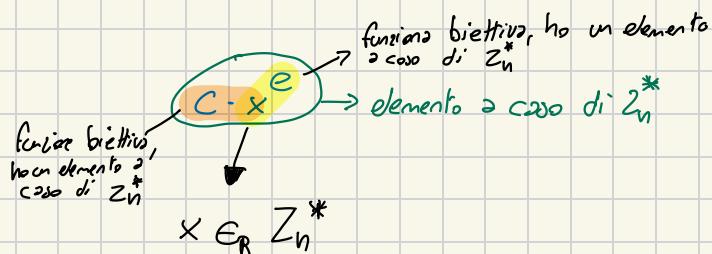
Piego a calcolare d? No poiché non posso calcolare
 $\varphi(n)$ da n poiché n è primo se e è n-1
ma altrimenti è $(p-1)(q-1)$ dove n = p·q e
quindi dovrei essere in grado di fattorizzare 1
primo primo

PUB KEY	n, e
PRIU KEY	n, d



è molto pericoloso perché l'algoritmo può calcolare la radice e extrarre di un numero

Sarebbe estremamente pericoloso perché in K^{-c} volte posso arrivare allo stesso

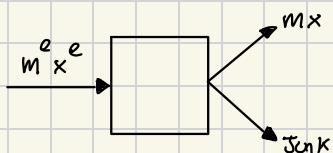


$$\sqrt[m]{C \cdot x^e} = \sqrt[m]{m^e \cdot x^e} = \frac{m \cdot x}{\cancel{x}} = m$$

Ciphertext
numero
 m^e

Dallo stesso si problema trasformato tras
lo stesso

Così facendo ho



Lezione 8

Non riuscire a decodificare

Se due agenti inviano allora i messaggi sono uguali?

HAMMELABLE: con due messaggi in chiaro, codifica legata

radice n-esima

non si usa il modolo, decodifichiamo facilmente

RSA, blocco di bit casuali + MEX, cipher block chain

Protocolli dimostrabilmente sicuri?

Nicchi, codifica di un singolo bit

Dispendioso

$$G: \mathbb{F}^K \mapsto (\mathcal{S}, \mathcal{P})$$

$$E: \mathcal{P}, m \mapsto E(m, P)$$

$$D: \mathcal{S}, m \mapsto D(c, s)$$

$$\forall m \quad D(E(m, P), s) = m$$

Algoritmo di Generazione

Scegliamo $n = p_1 p_2$ primi rappresentabili con $K/2$ bit $p_1, p_2 \in \mathbb{P}$ PRIMI
 $y \in \mathbb{F}_n$ non quadrato con $j=1$, scopro bit a caso

$$P_j = (m, y)$$

non quadrato
mi permette
di costruire

$$S_j = (p_1, p_2)$$

L'ipotesi di base è che sia difficile fattorizzare n , ma il problema di riferimento sarà: il problema del residuo quadratico, ovvero che il problema di calcolare la radice quadrata di un numero modulo n .

Encryption

$$b \mapsto \begin{cases} x^2 & b=0 \quad x \in \mathbb{F}_n \text{ QUADRATI} \\ x^2 y & b=1 \quad x \in \mathbb{F}_n \text{ QUADRATI} \end{cases}$$

per costruire un numero non quadrato casuale con simbolo di Jacobi 1, bisogna scegliere un numero casuale e verificare che il simbolo di Jacobi sia 1, ovvero che appartenga a \mathbb{Z}_n^*

verifica che il simbolo di Legendre rispetto a p_1 e q_1 sia -1

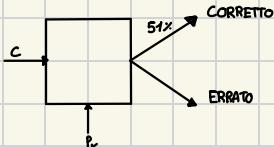
L'algoritmo di codifica prende un bit b , sia $x \in \mathbb{Z}_n^*$, $b=0$ allora $c=x^2 \pmod{n}$, altrimenti $c=x y \pmod{n}$, x^2 è un quadrato casuale di \mathbb{Z}_n^* , mentre $x y$ è un non quadrato con simbolo di Jacobi. Se prendo un quadrato con simbolo di Jacobi 1, è lo moltiplico per un non quadrato con simbolo di Jacobi -1. Ottengo un non quadrato con simbolo di Jacobi 1. Se il quadrato è casuale, allora otengo un non quadrato casuale con simbolo di Jacobi -1.

Il risultato è che la codifica di 0 è un quadrato a caso, mentre la codifica di 1 è un non quadrato a caso con simbolo di Jacobi -1.

COSA VUOL DIRE ATTACCARE IL PROTOCOLLO?

Decrittare è $1/2^k$, la probabilità di indovinare

Se riuscissi ad indovinare con probabilità leggermente superiore



il divario è maggiore con tanti esperimenti

se inferiore
lavoro il risultato

Algoritmo di decodifica

L'algoritmo di verifica prende in input c e verifica se c è un quadrato rispetto a p_1 e p_2 , allora $b=0$, se entrambe le verifiche falliscono allora $b=1$, in altri casi non siamo in presenza di un operante

$$\left(\frac{c}{p_1} \right) = \left(\frac{c}{p_2} \right) = 1 \quad \text{allora } b=0$$

$$\left(\frac{c}{p_1} \right) = \left(\frac{c}{p_2} \right) = -1 \quad \text{allora } b=1$$

$$\forall c \exists \bar{K} \forall K > \bar{K} \mid P[\text{successo}] - \frac{1}{2} > k^c$$

Trovare un numero che mi permette di alzare la probabilità di successo.

$P \mapsto$ prob. di successo

$$P[\text{MAGIORANZA DI EXP ABBAIA SUCCESSO}] = \sum_{i=\frac{n}{2}+1}^n P[X_i = \text{esp di successo}]$$

$$\sum_{i=\frac{n}{2}+1}^n \binom{n}{i} p^i (1-p)^{n-i}$$

$$\text{LIMITE DI CHERNOFF} \geq 1 - e^{-2n(p-\frac{1}{2})^2}$$

esponenzialmente piccolo in n

$\approx p \geq \frac{1}{2}$ mi avvicino a 1

Con un numero polinomiale di esperimenti mi avvicino a 1.

$$1 - e^{-2n(p-\frac{1}{2})^2} \geq 1 - 2^{-k}$$

$$e^{-2n(p-\frac{1}{2})^2} \leq 2^{-k}$$

$$-2n(p-\frac{1}{2})^2 \leq -2^{-k}$$

$$-2n(p-\frac{1}{2})^2 \leq -k$$

$$n \geq \frac{k}{2c(K-\frac{1}{2})^2} \quad \text{Quindi polinomiale}$$

Supponiamo che la probabilità di successo sia $\frac{1}{2} + k^c$

$$n \geq \frac{k}{2c(\frac{1}{2} + k^c - \frac{1}{2})} = \frac{k}{2c k^c} = \frac{1}{2c^2 k^c}$$

Attaccante con vantaggio polinomiale

Costriamo esperimenti indipendenti sull'analisi del vantaggio polinomiale

c_1 costrioso L esperimenti indipendenti

R_1, R_2, \dots, R_L numeri casuali in \mathbb{Z}_n^*

costrioso $CR_1^2, CR_2^2, \dots, CR_L^2 \leftarrow$ Input casuale, ho reso casuale il problema

Averemo stesso quadraticità di C , scegliere su un insieme più piccolo non funziona.

Supponiamo di avere una macchina che sui quadrati indovina al 40% non quadrati al 62%.

$$\text{Con } j=1 \quad \frac{1}{2} \cdot \frac{40}{100} + \frac{1}{2} \cdot \frac{62}{100} = \frac{102}{200} = 51\%$$

sbagliera' sempre perché se C è un quadrato a caso C deve essere distribuito come la macchina si aspetta

Allora la trasformazione non va bene, serie distribuzione uniforme su $j=1$.

Scegliamo anche bit a caso.

$b_1, b_2, \dots, b_j \leftarrow$ solo per aver distribuzione

Siano X_1, \dots, X_n variabili casuali e binarie indipendenti con probabilità di successo $P[X_i = 1] > \frac{1}{2}$ e $P[X_i = 0] = 1 - P[X_i = 1]$. La probabilità che più della metà delle variabili casuali siano 1 è:

$$P = \sum_{i=\frac{n}{2}+1}^n \binom{n}{i} p^{X_i=1} i p^{X_i=0}^{n-i}$$

$$P \geq 1 - e^{-2n(p-\frac{1}{2})^2}$$

Tale formula dice che la probabilità che più della metà degli eventi dia 1 è esponenzialmente vicina a 1, dove l'esponenzialmente è in funzione di n .

$$P[\text{errore}] = e^{-2n}$$

Dove E è il vantaggio

Supponiamo di volere $e^{-2n} < \frac{1}{2^k}$, quindi:

$$e^{-2cE^2 n} < 2^{-k}$$

$$-2cE^2 n < \frac{1}{2^k} - k$$

$$n > \frac{c^2 k}{2E^2}$$

Se E è polinomiale in K allora n è polinomiale in K . Di conseguenza, se il vantaggio è polinomiale in qualche security parameter, allora si riesce ad ottenere una quantità di errore nel security parameter che è esponenzialmente piccola, scegliendo una quantità di esperimenti polinomiale in E .

Un sistema è attaccabile nel momento in cui esiste un algoritmo polinomiale in grado di romperlo.

Nel momento in cui E è un K , allora n (dove n è il numero di esperimenti) è polinomiale in K .

Visto che l'ipotesi di partenza è che non esistono algoritmi probabilistici polinomiali in grado di rompere il sistema (vero) e visto che visto che abbiamo dimostrato che esiste tale algoritmo (falso), allora l'algoritmo di Micalli è sicuro.

$$y_i = \begin{cases} CR_i^2 & b_i = 0 \\ COR_i^2 & b_i = 1 \end{cases}$$

y non quadrato
con $j=1$

y_i è distribuito uniforme

Stiamo lanciando una moneta (b_i)
per decidere se invertire o meno la quadraticità.

Risponde al problema opposto, quindi lo complemento se b_i è 1

Prendo la maggior parte delle risposte

perché ho invertito quadraticità

Non siamo in grado di operare malleabilità poiché la chiave pubblica non ha y che essendo NON quadrato opererebbe il complemento del bit, ovvero l'unica operazione nota su un bit

Lezione 9

b_1, \dots, b_e

$E(b_1)E(b_2)\dots E(b_e)$

è un electronic code book in cui si compagno i singoli bit

NUFABILE passo scambiare i bit → proviamo a ignorare questo problema,
vorremmo che il ciphertext non si possa ridurre ad alcuna informazione binaria sul plaintext

Distinguisher D e PPT

Io distinguo due scenari diversi se in essi mi comporto in maniera diversa, restituisco 0 o 1 a seconda dello scenario che riconosco

per dire che un distinguisher non riconosce i vari scenari, vuol dire che si comporta identicamente in ciascuno di essi

E nasconde m_1, m_2 a D se

probabilità che il distinguisher riconosca
di distinguere m_1 e m_2 , m_1 dato il ciphertext
 m_2 dato il ciphertext con il security parameter che vale K

$$\forall k \in \mathbb{R} \quad |P_k^{D, m_1} - P_k^{D, m_2}| < K^{-c}$$

probabilità che il
distinguisher riconosca
 m_2 dato il ciphertext

Sto dicendo che non posso mai usare il distinguisher per capire se il ciphertext costituito è quello di m_1 o quello di m_2

Ovvero:

$$|P_k^{D, m_1} - P_k^{D, m_2}| < K^{-w(1)}$$

qualsiasi
misura
naturale

E nasconde a D informazioni se per ogni m_1 e m_2 E nasconde m_1 e m_2 a D $\{w, \{d\}\}, \dots$ quanti all'infinito
E nasconde se per ogni D e PPT E nasconde m_1 e m_2 a D

$\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}, \dots, w$

primo numero
infinito

"La nostra vita è
polinomialmente piccola"

$$\exists D \in PPT \quad \exists m_1, m_2 \quad \exists c \forall k \geq K \quad |P_k^{D, m_1} - P_k^{D, m_2}| \geq k^{-c}$$

Dimostrazione

Supponiamo che per assurdo $\exists D \in PPT$ tale che

$$\exists m_1, m_2 \quad \forall k \exists c \quad |P_k^{D, m_1} - P_k^{D, m_2}| > k^{-c}$$

Allora D puo' distinguere su due messaggi m_1 e m_2 che differiscono di un solo bit

$$m_1 = \alpha_1 \alpha_2 \dots \alpha_l$$

$$m_2 = \beta_1 \beta_2 \dots \beta_l$$

m_2 e' m_1 cambiando un bit alla volta

Definisco $\forall i \in \{0, \dots, l\}$ $m(i) = \beta_1 \dots \beta_i \alpha_{i+1} \dots \alpha_l$

$$m(0) = m_1 \quad P(0) = P_k^{D, m_1}$$

$$m(l) = m_2 \quad P(l) = P_k^{D, m_2}$$

$$P(i) = P_k^{D, m(i)}$$

ovvero la probabilita' che il distinguisher restituisca 1 se viene dato in input una codifica del messaggio m_i .

$$k^{-c} \leq |P(0) - P(l)| = \left| \sum_{i=0}^{l-1} P(i) - P(i+1) \right| \leq \sum_{i=0}^{l-1} |P(i) - P(i+1)|$$

Quindi:

$$\sum_{i=0}^{l-1} |P(i) - P(i+1)| \geq k^{-c}$$

diseguaglianza
di Weitzenböck

capacita' del
distinguisher
saper qualcosa

$$\frac{P(0)}{P(l)} - \frac{P(l)}{P(0)}$$

\hookrightarrow Sappiamo che

resta $P(0) - P(l)$

$$\frac{P(l)}{P(0)} - \frac{P(0)}{P(l)}$$

Avevo la somma di numeri che eccede un determinato valore allora so che esiste almeno un elemento che e' maggiore o uguale della media

$$\exists i \in \{0, \dots, l-1\} \text{ t.c. } |P(i) - P(i+1)| \geq \frac{k^{-c}}{l} > k^{-c}$$

Stiamo distinguendo due messaggi che differiscono per un singolo bit, e abbiamo trovato un pariano c'

$$m(i) = \beta_1 \dots \beta_i \boxed{\alpha_{i+1}} \alpha_{i+2} \dots \alpha_l$$

$$m(i+1) = \beta_1 \dots \beta_i \beta_{i+2} \alpha_{i+3} \dots \alpha_l$$

Supponiamo senza perdita di generalita' che $\alpha_{i+1} = 0$ e $\beta_{i+1} = 1$. Sia z un elemento di \mathbb{Z}_n^* con $(\frac{z}{n}) = 1$, quindi z potrebbe essere la codifica di uno 0 o di un 1. Data z viene costruito $E(\beta_1)E(\beta_2) \dots E(\beta_i)zE(\alpha_{i+2}) \dots E(\alpha_l)$ e viene lanciato l'algoritmo D sul risultato. La probabilita' con cui D restituisce 1 e' $P(i)$ se z codifica 0 e $P(i+1)$ se z codifica 1. Se in input viene data la codifica $m(i)$ secondo l'algoritmo per codificare il messaggio m_1 , ovvero applicando E a tutti i bit del messaggio. Avendo però aggiunto z in mezzo al messaggio non ha codificato secondo l'algoritmo che il distinguisher si aspetta, quindi bisogna codificare il messaggio $m(i)$ in maniera corretta:

$$E(\beta_1)E(\beta_2) \dots E(\beta_i) (\boxed{z}) E(\alpha_{i+2}) \dots E(\alpha_l) \quad \text{con } z \in \mathbb{Z}_n^*$$

$z \in \mathbb{Z}_n^*$
casuale

A questo punto il distinguisher restituisce 1 con probabilita' $P(i)$ se z codifica 0 e $P(i+1)$ se z .

Chiamiamo P_0 la probabilita' che il distinguisher restituisca 0 (quindi $P(i)$) e P_1 la probabilita' che il distinguisher restituisca 1 (ovvero $P(i+1)$) se in input dato e' costituito come sopra

Se si ricava la codifica di un bit scelto a caso, con quale probabilita' si riesce a distinguere se $z = 0$ o 1 ? Si utilizza il risultato del distinguisher D come testistivo

$$P[\text{individuare}] = \frac{1}{2} \cdot (1 - P_0) + \frac{1}{2} \cdot P_1 = \frac{1}{2} + \frac{1}{2} (P_1 - P_0)$$

$$\text{Se sottraiamo } \frac{1}{2} \text{ otteniamo: } P[\text{non individuare}] = \frac{1}{2} + \frac{1}{2} \cdot (P_0 - P_1) - \frac{1}{2}$$

$$= \frac{1}{2} (P_0 - P_1)$$

$$= \frac{1}{2} \cdot |P_0 - P_1|$$

$$\geq \frac{1}{2} \cdot k^{-c'}$$

\rightarrow L'algoritmo ha quindi un vantaggio di almeno $\frac{1}{2} \cdot k^{-c'}$

e quindi tale algoritmo e' un attaccante.

ho costruito un distinguisher tra quadrati e non quadrati

$$|P_k^{D, 0} - P_k^{D, 1}| > k^{-c'}$$

Distinguisher

$$|P_k^{D, 0} - P_k^{D, 1}| > k^{-c'}$$

$$|P[b \in_R \{0, 1\}; e = E(b); b' = A(e); b == b'] - \frac{1}{2}| > k^{-c'}$$

Lezione 10

Abbiamo il nostro algoritmo A che indovina

$$A \quad \left| P_e [b \in \{0,1\}; c \leftarrow E(b); b' = A(c); b = b'] - \frac{1}{2} \right| > k^{-c}$$

II

$$D \quad \left| P_k^{D,0} - P_k^{D,1} \right| > k^{-c}$$

$$P_k^{D,0} = P_e [c \leftarrow E(0); A(c) == 1]$$

$$P_k^{D,1} = P_e [c \leftarrow E(1), A(c) == 1]$$

$$P_e[x] = P_e[b=0]P_e[x|b=0] + P_e[b=1]P_e[x|b=1]$$

$$= \frac{1}{2} (1 - P_k^{D,0}) + \frac{1}{2} P_k^{D,1}$$

Supponiamo di indovinare

$$P_e[x] > \frac{1}{2} + k^{-c} \quad \text{without loss of generality}$$

$$\frac{1}{2} (1 - P_k^{D,0}) + \frac{1}{2} P_k^{D,1} > \frac{1}{2} + k^{-c}$$

$$\frac{1}{2} - \frac{1}{2} P_k^{D,0} + \frac{1}{2} P_k^{D,1} > \frac{1}{2} + k^{-c}$$

$$\frac{1}{2} (P_k^{D,1} - P_k^{D,0}) > k^{-c}$$

$$P_k^{D,1} - P_k^{D,0} > 2k^{-c} > k^{-c}$$

$$\frac{1}{2} (1 - P_k^{D,0}) + \frac{1}{2} P_k^{D,1} = \frac{1}{2} - \frac{1}{2} P_k^{D,0} + \frac{1}{2} P_k^{D,1} = \frac{1}{2} + \frac{1}{2} (-P_k^{D,0} + P_k^{D,1})$$

$$> \frac{1}{2} - \frac{1}{2} k^{-c} > \frac{1}{2} + k^{-c}$$

Costruire un distinguisher è equivalente a costruire qualcosa che indovina

Abbiamo costruito il primo critosistema dimostrabilmente sicuro

Il lato negativo è che costa K bit (oggi K=1000) ogni bit cifrato

↳ è un problema, sprecando banda e risorse → vorremmo codificare un bit con n solo bit → impareremo a costruire numeri pseudo-caos di entropia sufficientemente sicuri da utilizzare come base per stabilire one time pad (fino' da chiave)

Lancio della moneta in rete

→ problema difficile da simulare

↓
nessuno con potenza di calcolo polinomiale può notare la differenza

Abbiamo due agenti: A, B

COIN_A, COIN_B

A, B ONESTI

$$\text{COIN}_A = \text{COIN}_B \quad P_e[\text{COIN}_A = 0] = \frac{1}{2}$$

A ONESTA, B DISONESTO

$$\left| P_e[\text{COIN}_A = 0] - \frac{1}{2} \right| < k^{-w(d)}$$

A DISONESTA, B ONESTO

$$\left| P_e[\text{COIN}_B = 0] - \frac{1}{2} \right| < k^{-w(d)} \rightarrow \text{discordanza minore di qualsiasi polinomio}$$

per ogni esponente esiste un valore minimo della lunghezza della chiave tale per cui "chiavi" sufficientemente lunghe, la probabilità si discosta da $\frac{1}{2}$ di un valore minore di $k^{-w(d)}$

Affidare il lancio a un terzo agente porta con sé la medesima problematica: l'agente deve essere fidato

Lancio della moneta con residuo quadratico

Ci basiamo sulla difficoltà di stabilire se un numero è quadratico o meno in \mathbb{Z}_p^*

A p,q primi a caso

$$n = p \cdot q$$

$$\mathbb{Z}_n \text{ con } \left(\frac{\cdot}{n}\right) = 1$$

A B



$$b \in \mathbb{Z}_{n-1}$$

$$\text{COIN}_A = \text{COIN}_B = b \odot \text{IS_SQUARE}(z)$$

$$\downarrow$$

$$\frac{1}{2}$$

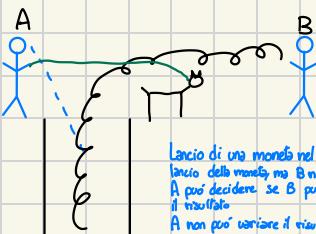
Se Alice sa qual è il risultato favorevole a lei e Bob non lo sa, Alice quando ottiene un messaggio negativo per lei non risponde.

Alice ha quindi la capacità di alterare la probabilità.

Il protocollo non è sicuro.

Siamo in grado di dimostrare che non esiste alcun protocollo in grado di garantire quella probabilità.

Lancio di monete nel pozzo



Lancio di una moneta nel pozzo; A vede il risultato del lancio della moneta ma B no.

A può decidere se B può avvicinarsi o meno a vedere il risultato.

A non può variare il risultato del lancio della moneta, può solo impedire a B di vedere il risultato.

Se è B a decidere se partecipa o meno al protocollo, non gli interessa il risultato del lancio della moneta, ma interessa solamente che la probabilità del lancio sia almeno $\frac{1}{2}$.

Sostanzialmente: A lancia una moneta (sceglie a caso fra un quadratico e un non quadratico) e B sceglie un bit B è caso e lo invia ad A. Quando B riceve z, non so quel è il risultato del lancio della moneta, perché non so risolvere il problema del residuo quadratico, di conseguenza è vero che A invia prima il risultato del proprio lancio della moneta, ma nella condizione in cui B non vede il risultato, ma B non invia il bit b in funzione di z?

Se entrambi gli agenti sono onesti, e' stato campionato secondo una misura casuale, quindi z è un quadratico con probabilità $\frac{1}{2}$ e non quadratico con probabilità $\frac{1}{2}$. Anche b è stato campionato secondo una misura casuale, quindi b è 0 con probabilità $\frac{1}{2}$ e 1 con probabilità $\frac{1}{2}$. Alla fine del protocollo, tutti sono in grado di calcolare il risultato perché tutti conoscono gli altri parametri dell'algoritmo, che sarà ugale per entrambi seguendo la formula: $b \odot \text{isSquare}(z)$

Se B è disonesto, allora può calcolare bit b non casuale, distribuendolo in maniera differente. Se B calcola b indipendentemente da z, allora qualunque sia la regola di calcolo di b, sarà sempre uguale allo quadratico/bi di z con probabilità $\frac{1}{2}$. Un altro modo per imbrogliare è quello di calcolare b in funzione di z, ma questo richiede la conoscenza dell'algoritmo della quadraticità di z. B ha il vantaggio di allontanarsi da $\frac{1}{2}$, ma è il vantaggio pari a quello della risoluzione del problema del residuo quadratico, quindi più piccolo di qualsiasi polinomio.

Scegliendo uniformemente tra gli elementi di \mathbb{Z}_n^* con simbolo di Jacobi 1, si ottiene un quadratico con probabilità $\frac{1}{2}$

Se A è disonesto, può calcolare p e q non primi, ma B, alla fine del protocollo se ne accorge ricevendo p e q e quindi potrebbe lanciare una moneta una propria moneta per decidere il proprio risultato o se, in occasioni del risultato che gli è sfavorevole, può decidere come risultato un valore che gli fornirebbe tale vantaggio. Non potrebbe non inviare p e q, in questo caso B ha un enorme vantaggio, perché dopo il secondo messaggio A conosce coin, ma B non conosce coin, se A invia il proprio messaggio a B, B è in grado di calcolare COIN e quindi il risultato finale, ma se A lo fa il risultato vincente e il risultato offerto è sfavorevole, A non spedisce il terzo messaggio e B lancia la propria moneta che con probabilità $\frac{1}{2}$ è sfavorevole, ma con probabilità $\frac{1}{2}$ è sfavorevole, si crea una situazione sfavorevole a B.

$$P[\text{COIN}_B = 0] = \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2} = \frac{3}{4}$$

Il comportamento di A ha alterato la misura di probabilità di successo di B a favore di A.

Il protocollo appena descritto non soddisfa le proprietà di un protocollo di lancio della moneta. Ci sono casi in cui il protocollo però può essere seguito, ovvero quando chi impersona A non conosce il risultato e B può o non può conoscere il risultato vincente.

Un protocollo per funzionare deve essere tale che gli agenti possono conoscere il risultato anche senza l'ultimo messaggio. Se l'ultimo messaggio è quello significativo, allora il protocollo non è corretto.

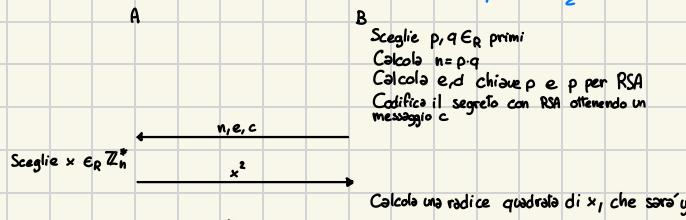
Sia n il numero minimo di messaggi di un protocollo che funziona, quindi gli agenti conoscono il risultato del lancio della moneta anche senza l'ultimo dell'ultimo messaggio. Cio vuol dire che l'ultimo messaggio non serve, ma se non serve allora il penultimo messaggio è quello che permette ad A di conoscere il risultato, ma se B decide di non inviare tale messaggio si imposta una reazione a catena che

Non esiste il protocollo per il lancio della moneta che soddisfi le condizioni dette da noi. Il protocollo funziona solo se, con n agenti, ad imbrogliare sono meno di $\frac{n}{2}$ agenti.

Oblivious Transfer

Supponiamo che B sappia il risultato della borsa del prossimo anno e che chieda ad A di avere in cambio del denaro. B non vuole dare tutto il denaro ad A, ma solo una parte. A questo punto A decide di fare il lancio della moneta per decidere se mostrare o meno il risultato della borsa a B. A questo punto deve trasferire l'informazione ad A, ma non gli interessa se l'informazione è stata trasferita o meno, ma solamente che la probabilità che l'informazione sia stata trasferita sia di almeno $\frac{1}{2}$

Quindi la fattorizzazione di n deve essere trasmessa con probabilità $\frac{1}{2}$



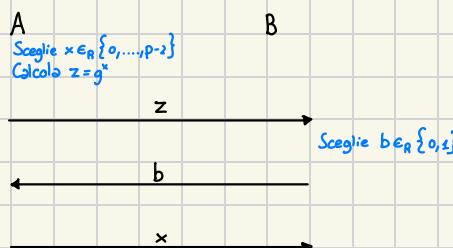
Quindi A invia un quadrato a caso e B invia una delle quattro radici quadrate di x. Con probabilità $\frac{1}{2}$ tale radice sarà diversa da $\pm x$. Quando un agente possiede due radici quadrate di uno stesso numero che non sono una l'opposto dell'altro, allora è in grado di fattorizzarne n. Con probabilità $\frac{1}{2}$, B invierà ad A una radice diversa da $\pm x$, metterà quindi A nelle condizioni di calcolare p e q e quindi d per decodificare il messaggio c. B non ha alcun interesse nel risultato che otterrà con probabilità $\frac{1}{2}$, ma interessa che A sia riuscito a decifrare c con probabilità $\frac{1}{2}$.

Il vantaggio di A è implicito nel risultato

Lancio della moneta con il logaritmo discreto

Il problema è che fino ad ora abbiamo lavorato con la "quadraticità" di un numero in \mathbb{Z}_n^* , ma vorremmo lavorare con singoli bit. Quindi non con l'inversa di una funzione one way, ma con singoli bit, ovvero con l'informazione binaria dell'inversa di una funzione one way.

Disponiamo di un numero p e un generatore g di \mathbb{Z}_p^* , quindi g è un numero che genera tutti gli elementi di \mathbb{Z}_p^* , ovvero $\{g^0, g^1, \dots, g^{p-1}\}$. I due valori p e q sono condivisi tra A e B, prima dell'inizio del protocollo.



Il risultato è quindi $b \oplus (x < \frac{p-1}{2})$. L'idea di fondo è che B quando sceglie b non è in grado di decidere il risultato del lancio della moneta, perché dovrebbe verificare se x è minore o meno di $\frac{p-1}{2}$, facendo quindi un test binario sul logaritmo discreto di z. Nel momento in cui A invia x, B è in grado di verificare la correttezza del risultato e a calcolare il risultato.

B non è in grado di calcolare il predicato binario $x < \frac{p-1}{2}$, a meno di un vantaggio più piccolo di qualsiasi polinomio

Tale algoritmo utilizza un predicato binario sull'inversa di una funzione one way.

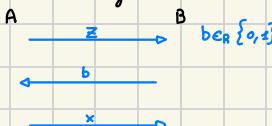
Dato una funzione one way f, un predicato binario P sull'inversa di f, non siamo certi che il predicato sia difficile da calcolare.

Per il logaritmo discritto, il predicato binario facile da calcolare è il bit meno significativo di x. Tale bit vale zero se, e solo se, z è un quadrato in \mathbb{Z}_p^* . Ma tale predicato dice qualcosa di differente, ovvero ci chiediamo se il logaritmo discreto di z sia nella prima metà o nella seconda metà dei logaritmi discreti di \mathbb{Z}_p^* o nella seconda, ovvero una sorta di bit più significativo di x.

P, g

A sceglie $x \in_R \{0, \dots, p-1\}$

$$z \leftarrow g^x$$



$$b \oplus \left(x < \frac{p-1}{2} \right)$$

HARD CORE PREDICATE

Qualunque funzione one way ha un hard core predicate. Si tratta di un predicato binario sull'inversa di una funzione one way, che è difficile da calcolare. Si scegliesse a caso il valore di f, tale predicato binario, lo probabilità con cui un algoritmo riesce a calcolare il predicato conoscendo z e $\frac{p-1}{2}$, con E più piccolo di qualsiasi polinomio.

Lezione 11

Hardcore predicate del logaritmo discreto

Sia $y = g^x$ e sia il predicato in considerazione $x < \frac{p-1}{2}$. Se $y \in \mathbb{Z}_p$ allora ammette due radici quadrate, z_1 e z_2 , di queste due radici, una ha logaritmo discreto minore di $\frac{p-1}{2}$ e l'altra maggiore o uguale a $\frac{p-1}{2}$. Questo perché le rispettive radici quadrate sono nella forma g^i e $g^{i+\frac{p-1}{2}}$. La radice quadrata nella forma g^i è chiamata radice quadrata principale.

Di base, se abbiamo due radici quadrate di y , che sappiamo calcolare aritmeticamente, non sappiamo come capire quale sia la radice quadrata principale, ma se avessimo a disposizione l'algoritmo per calcolare se il logaritmo discreto di un numero è minore di $\frac{p-1}{2}$, allora potremmo calcolare la radice quadrata principale di un quadrato.

Supponiamo di avere un logaritmo che date le due radici quadrate di un numero distingue la radice quadrata principale. In questo caso possiamo costruire un algoritmo che calcola il predicato binario $x < \frac{p-1}{2}$.

TEOREMA

Se esiste un algoritmo per il calcolo della radice quadrata principale di un quadrato in \mathbb{Z}_p^\times , allora esiste un algoritmo efficiente per il calcolo del logaritmo discreto in \mathbb{Z}_p^\times .

TEOREMA

Se esiste un algoritmo che riesce a calcolare il predicato binario con vantaggio polinomiale rispetto al caso casuale, allora esiste un algoritmo probabilistico polinomiale per il calcolo del logaritmo discreto.

Per dimostrare il teorema doveremo seguire tre passaggi:

1. Mostriamo che siamo in grado di calcolare il logaritmo discreto di un numero avendo in mano un algoritmo per il calcolo della radice quadrata principale di un quadrato
2. Mostriamo che tale algoritmo funziona anche se la radice quadrata principale è calcolata con probabilità esponenzialmente vicina a 1
3. Mostriamo che, partendo da un algoritmo che funziona con vantaggio polinomiale rispetto al caso casuale, siamo in grado di costruire un algoritmo che funziona con probabilità esponenzialmente vicina a 1.

Calcolare il logaritmo discreto avendo a disposizione l'algoritmo per il calcolo della radice quadrata principale (PSQR)

Autremo è disponibile l'algoritmo PSQR che calcola la radice quadrata principale di un quadrato in \mathbb{Z}_p^\times , avendo a disposizione il predicato binario. Per farlo prendere in input y e calcola la radice quadrata, verifica il predicato binario per capire se è la radice quadrata principale, se lo verifica fallisce, calcola la radice principale opposta e restituisce il risultato.

procedure $LSB(y)$

```
if  $\frac{y}{p} = 0$  then
    return 0
else
    return 1
end if
```

end procedure

Idea: abbiamo macchina B



$$\geq \frac{1}{2} + \frac{c}{2}$$

$$\sqrt{y} \quad \boxed{g^{\frac{x}{2}}} \quad g^{\frac{x-p-1}{2}}$$

Ricordiamo che la procedura LSB verifica se il bit meno significativo di un numero è pari o dispari. Se è pari restituisce 0, altrimenti restituisce 1.

procedure $DISCRETELOGARITHM(y)$

```
if  $y=1$  then
    return 0
end if
b ←  $LSB(\text{DiscreteLogarithm}(y))$ 
if  $b=1$  then
    return  $y \leftarrow y \cdot g^{-1}$  // imposta a zero il bit meno significativo
end if
 $y \leftarrow PSQR(y)$  // scorsi a destra di un bit
return  $2 \cdot \text{DiscreteLogarithm}(y) + b$ 
end procedure
```

Per calcolare la radice quadrata abbiamo necessariamente bisogno del bit meno significativo a zero. L'algoritmo ricorsivo permette di calcolare il logaritmo discreto di un numero calcolando il bit meno significativo e riconducendo il calcolo dello stesso problema in una situazione in cui si ha un bit in meno.

Se abbiamo un algoritmo che funziona per il calcolo della radice quadrata principale allora possiamo costruire un algoritmo che calcola il logaritmo discreto.

Supponiamo di avere un algoritmo B per PSQR che funziona con probabilità esponenzialmente vicina a 1, ovvero $1-\epsilon$. La probabilità che l'algoritmo DiscreteLogarithm invocato K volte fornисca sempre la risposta corretta è $(1-\epsilon)^K$. L'algoritmo Discrete Logarithm può non funzionare, ma dobbiamo modo di accorgerci:

- Se il numero di passi è maggiore del numero di bit di y , allora siamo sicuri che il PSQR non ha funzionato.
- Sul risultato finale possiamo verificare che $g^x = y$. Se non è così allora PSQR non ha funzionato.

In media se funziona con probabilità $(1-\epsilon)^K$, allora il numero di volte che funziona in cui si dovrà lanciare l'algoritmo sarà $\frac{1}{(1-\epsilon)^K}$. Se è $\leq \frac{1}{2}$, allora $(1-\epsilon)^K \geq \frac{1}{2^n}$.

Se abbiamo un algoritmo che funziona con probabilità esponenzialmente vicina a $\frac{1}{2}$, allora possiamo costruire un algoritmo che calcola il logaritmo discreto con probabilità almeno $\frac{1}{2}$, quindi se riusciamo in media due volte l'algoritmo ottieniamo la risposta corretta. Questo perché possiamo verificare che il risultato sia corretto.

Riusciamo ora a mostrare che se ci viene dato un algoritmo che funziona con vantaggio polinomiale rispetto a $\frac{1}{2}$, allora possiamo costruire un algoritmo che funziona con probabilità esponenzialmente vicina a $\frac{1}{2}$.

Analiticamente sì, per fatto dobbiamo costruire tante istanze indipendenti dello stesso problema e prendendo il risultato assurso la maggior parte delle volte. Come costruiamo gli esperimenti indipendenti, tali per cui nel momento in cui conosciamo la risposta al problema costruito, allora troviamo la risposta al problema originale?

Costruzione dell'esperimento indipendente g^r tale per cui dalla risposta g^r ottieniamo la risposta a y .

TEOREMA

Sia y un quadrato in \mathbb{Z}_p^* e sia $r \in [0, \dots, \frac{p-1}{2}]$ (esponente a caso per una possibile radice quadrata di un numero, presente nella prima metà). Sia $y = g^{2r}$, dove y sarà un'altra istanza indipendente del problema dei residui quadratici, distribuiti uniformemente in \mathbb{Z}_p^* , di cui conosciamo il logaritmo discreto. Se $2x+2r < p-1$ allora $z = g^r$ è PSQR di y ; g^r dà z è PSQR di y .

$$\text{Sappiamo che } \sqrt{g^{2r} g^{2r}} = \begin{cases} g^{2r} \\ g^{2r} \end{cases}$$

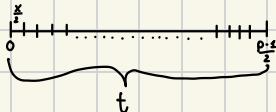
Con l'ipotesi $2x+2r < p-1$ allora g^{2r} è la radice quadrata principale

Questo perché moltiplicando $g^{2r} g^{2r}$ abbiamo un'operazione aritmetica poiché gli esponenti sono sommati e non si trova nella radice quadrata successiva, cuore quella dopo la prima metà (l'esponente di y è $2x$). Moltiplichiamo quindi g^r che non si vede oltre $p-1$, quindi il logaritmo discreto del prodotto è la somma dei logaritmi discriti; andar oltre $p-1$ significa fare un'indietro, siccome siamo in un gruppo ciclico.

Dalla risposta al problema trasformato, sappiamo quindi calcolare la risposta al problema originale.

Ma come faccio a far sì che r scelto casualmente soddisfi le proprietà del teorema?

Più x è piccolo e più è probabile trovare un r che soddisfa la proprietà del teorema



Supponiamo che $\frac{t}{2}$ sia nel primo intervallo, la probabilità di trovare un r dove $\frac{x}{2} + r$ non vada oltre $\frac{p-1}{2}$, cuore $\frac{t-1}{2}$. La probabilità che la risposta sia corretta è la combinazione delle probabilità di soddisficiabilità del teorema, cuore $(\frac{t-1}{2})$ e la probabilità della risposta al problema trasformato, cuore $(\frac{1}{2} + k^{-c})$.

Questo perché qualche volta, dalla risposta al problema trasformato, non si riesce a trovare la risposta al problema originale.

$$P[\text{Esperimento corretto}] = \left(\frac{t-1}{t}\right) \left(\frac{1}{2} + k^{-c}\right) \geq \frac{1}{2} + k^{-c}$$

Risolvendo la disequazione in t riusciamo a capire quale è il limite inferiore e gli intervalli da utilizzare.

Dalla teoria dell'algebra sappiamo che si tratta di una disequazione polinomiale in t , come tale ha una soluzione polinomiale nelle costanti interne. La soluzione è quindi polinomiale nelle costanti presenti nell'equazione

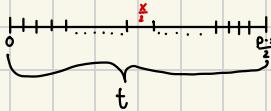
$$\frac{t-1}{t} \geq \frac{1}{2} + k^{-c} \Rightarrow t \geq \frac{2}{1+2k^{-c}}$$

Dividendo in una "quantità" di intervalli polinomiale in K , otteniamo un sistema tale per cui, se x è nel primo intervallo, la "probabilità" di successo, ovvero di fornire una radice quadrata principale è polinomialmente distante da $\frac{1}{2}$, di conseguenza abbiamo un algoritmo che risolve il problema in tempo polinomiale.

Abbiamo quindi costruito un algoritmo che con probabilità $\frac{1}{2}$ calcola il logaritmo discreto di un numero, a patto che la metà di tale logaritmo stia nel primo intervallo.

Generalizzazione dell'intervallo

Supponiamo di sapere che $\frac{x}{2}$ sia nell' i -esimo intervallo.



Sappiamo che $y = g^x$ e che $i \frac{P-1}{2} \leq \frac{x}{2} < (i+1) \frac{P-1}{2}$

$$g^{x'} = y \cdot g^{\frac{P-1}{2}} \Rightarrow g^{x'} = g^{\frac{x}{2} - \frac{P-1}{2} i} \Rightarrow x' = x - \frac{P-1}{2} i$$

Quindi x' sia nel primo intervallo, infatti da $\frac{x}{2}$ abbiamo tolto il punto d'inizio dell'intervallo i -esimo, togliendo tale quantità abbiamo ottenuto il risultato corrispondente al primo intervallo. Ci siamo quindi ricondotti alla risoluzione di un problema relativo al calcolo della radice quadrata principale, ovvero il problema che abbiamo già risolto. Dopo aver risolto il problema ci riconduciamo nuovamente al problema originale, moltiplicando per $g^{\frac{P-1}{2} i}$.

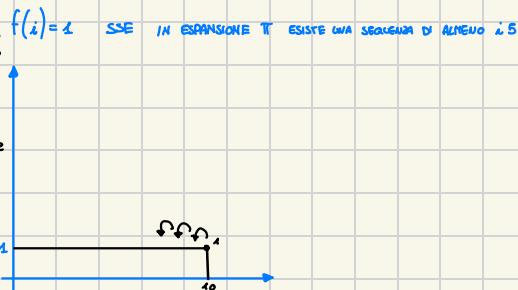
Non conoscendo l'intervallo in cui si trova $\frac{x}{2}$, possiamo provare a risolvere il problema per tutti gli intervalli. Essendo quantità polinomiali in K .

Lezione 12

Vediamo costruire una sequenza di bit che sia equivalente ad un lancio di monete. Se vogliamo utilizzare all'interno di un programma dei bit che sono casuali, saremo bisogno di un processo di lancio di monete all'interno del nostro calcolatore. All'interno del processo di lancio di monete utilizzato c'è sempre un punto in cui ci sono scelte casuali da dover effettuare e tali scelte possono essere fatte da un processo realmente casuale. All'interno di un calcolatore non abbiamo un processo realmente casuale, poiché i calcoli e le scelte sono deterministici. Il processo non è casuale, ma deterministico; ma appena come processo casuale agli occhi di chi lo osserva.

L'algoritmo deterministico esegue un'operazione sul seme e produce un output. La generazione non può essere tale che la distribuzione di probabilità degli elementi sia uniforme, ma deve essere tale che produca una sequenza di bit tale per cui nessuna con polinomi di calcolo probabilistica polinomiale sia in grado di capire la distribuzione. Chiunque non si deve accorgere del fatto che non stiamo lavorando con sequenze casuali, ma con sequenze pseudocasuali.

Generazione Sequenze Bit Pseudocasuali PRSG



Il seme s viene scelto in maniera realmente casuale ed è corso ed ha K bit. E l'output ha l bit, normalmente con $l > K$. I generatori di numeri pseudocasuali possiamo vederlo come un moltiplicatore di casualità: Nessun algoritmo probabilistico polinomiale sarà in grado di trovare una regolarità all'interno dell'output generato, normalmente ci sarà.

La probabilità che qualsiasi algoritmo sia in grado di predire il bit b_{D+1} avendo la sequenza b_0, \dots, b_D si discute da $\frac{1}{2}$ di una quantità più piccola di qualsiasi polinomio

$$\textcircled{1} \quad \forall_{\lambda} \forall_{c} \exists_{k} \forall_{K} \forall_{n} \left[P_e [S \in \{0,1\}^K; b_0, b_1, \dots, b_2, b_{D+1} = G(s); b = A(b_0, \dots, b_D); b = b_{D+1}] = \frac{1}{2} < \frac{c}{K^{\lambda}} \right] = \frac{1}{K^{-\lambda}}$$

Una sequenza di bit è pseudocasuale se nessun algoritmo probabilistico polinomiale è in grado di distinguere la sequenza generata da una sequenza realmente casuale

La probabilità che l'algoritmo D dia come risultato 1, avendo che abbia riconosciuto la sequenza pseudocasuale come tale:

$$P_k^{D,G} = P_e [s \in \{0,1\}^K; b_0, b_1, \dots, b_D \leftarrow G(s); D(b_0, \dots, b_D) = 1]$$

Descriuiamo la probabilità che l'algoritmo D dia come risultato 1, se prende in input una sequenza di bit realmente casuale

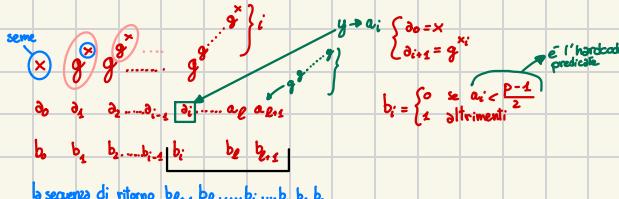
$$P_k^{D,R} = P_e [b_0, b_1, \dots, b_D \in \{0,1\}^K; D(b_0, \dots, b_D) = 1]$$

Quindi la probabilità che il distinguere realistico 1, nei due casi differisce di una quantità più piccola di qualsiasi polinomio

$$\textcircled{2} \quad \forall_{\lambda} \forall_{c} \left| P_k^{D,G} - P_k^{D,R} \right| < \frac{c}{K^{-\lambda}}$$

① e ② sono equivalenti, cioè ci torna utile perché è possibile utilizzare la definizione più comoda

Fissiamo un numero primo p e un generatore g del gruppo \mathbb{Z}_p^* . Prendiamo il seme $x \in \{0, \dots, p-1\}$ e definiamo:



Utilizzando la definizione 3.8, supponiamo per assurdo che esista un algoritmo in grado di predire il bit successivo con un vantaggio più grande di qualche polinomio. I bit presi in input sono b_0, \dots, b_D , quindi l'algoritmo A predice il bit b_{D+1} . Ma a_3 è l'hardcore predice di b_2 , quindi non è possibile predire il bit b_2 con un vantaggio maggiore di qualche polinomio

$$a_3 = g^{b_2} \text{ e } a_2 = g^{b_1}$$

La costruzione della sequenza $a_0, a_1, a_2, a_3, \dots, a_D$ e della sequenza $b_0, b_1, b_2, \dots, b_{D+1}$ è facilmente costituibile. Se i bit fossero stati restituiti in ordine sequenziale $b_0, b_1, b_2, \dots, b_D$, non saremmo riusciti a far funzionare l'algoritmo perché i bit successivi della sequenza sarebbero stati il risultato di una funzione facilmente calcolabile. La nostra costruzione si è basata sul fatto che la funzione da calcolare sui semi è difficile al fine di dire che non riusciamo a prevedere il bit successivo

Il fatto che la dimostrazione non fornisce non implica che non sia possibile restituire i bit in ordine di enumerazione crescente

La costruzione di funzionamento con i bit restituiti alla rovescia si basa sul fatto che conoscendo i semi non siamo in grado di calcolare b_2 , se non so calcolare b_2 conoscendo i semi a maggior regole non sono in grado di calcolare conoscendo i bit, ma tale dimostrazione non mi permette di raffutare tale concetto.

Prendendo in considerazione la formula 3.9, se i bit restituiti alla rovescia sono indistinguibili da una sequenza casuale, allora anche i bit restituiti in ordine crescente sono indistinguibili da una sequenza casuale.

Se b_0, \dots, b_n soddisfa la definizione 3.9 allora soddisfa la definizione 3.8. Quindi b_0, \dots, b_n è indistinguibile da una sequenza casuale e possiamo concludere che b_0, \dots, b_n è indistinguibile da una sequenza casuale.

Sia D un distinguisher per b_0, \dots, b_n , quindi D prende in input b_0, \dots, b_n e restituisce 1 se la sequenza è pseudocasuale e 0 se la sequenza è casuale. Sia D' un distinguisher per b_0, \dots, b_n , quindi D' prende in input b_0, \dots, b_n e restituisce $D(b_0, \dots, b_n)$, quindi rovescia la sequenza e restituisce il risultato di D basato sulla sequenza rovesciata.

Se D distingue b_0, \dots, b_n allora D' distingue b_n, \dots, b_0 e viceversa. Quindi distinguere i bit in ordine di enumerazione crescente è equivalente a distinguere i bit in ordine di enumerazione decrescente.

Restituire i bit alla rovescia è scatalogico perché non è possibile calcolare il bit successivo, mentre restituire i bit in ordine di enumerazione crescente permette di andare avanti con la sequenza.

Distingere equivale a predire.

TEOREMA

1. Sia distinguere equivalente a:

$$V_{D \text{ PRED}} |P_k^{D,G} - P_k^{D,R}| < k^{-\omega(\epsilon)}$$

2. Sia prevedere equivalente a:

$$|P[\exists s \in_R \{0,1\}^k : b_0 b_1 \dots b_{2s-2} \leftarrow A(b_0 b_1 \dots b_{2s-2})_2 = b_{2s+1}] - \frac{1}{2}| \leq k^{-\omega(\epsilon)}$$

Allora:

$$\cdot 1 \Rightarrow 2 \quad \epsilon = 2 \Rightarrow 1$$

Dimostrazione ($1 \Rightarrow 2$): Su input b_0, \dots, b_{2s-2} indoviniamo b_{2s} con vantaggio polinomiale, ovvero:

$$P[A(b_0, \dots, b_{2s-2}) = b_{2s}] - \frac{1}{2} \geq k^{-c}$$

$$P[A(b_0, \dots, b_{2s-2}) = b_{2s}] \geq \frac{1}{2} + k^{-c}$$

e possiamo affermare senza perdita di generalità: Se vogliamo dimostrare che la proprietà (2) vale, ovvero che violare la proprietà (1) si viola anche la proprietà (2), allora possiamo costruire un distinguisher.

$$D(b_0, \dots, b_{2s}) = \begin{cases} 1 & \text{se } A(b_0, \dots, b_{2s-2}) = b_{2s} \\ 0 & \text{altrimenti} \end{cases}$$

Sappiamo che la probabilità che l'algoritmo A indovini un bit scelto in maniera veramente casuale e indipendente da b_0, \dots, b_{2s-2} è esattamente $\frac{1}{2}$ (one time pad) ovvero:

$$P_k^{D,U} = \frac{1}{2}$$

Mentre la probabilità che A , con input b_0, \dots, b_{2s-2} , indovini b_{2s} è maggiore di $\frac{1}{2} + k^{-c}$, ovvero:

$$P_k^{D,G} \geq \frac{1}{2} + k^{-c}$$

Di conseguenza:

$$P_k^{D,G} - P_k^{D,U} \geq \frac{1}{2} + k^{-c} - \frac{1}{2} = k^{-c}$$

Se esiste l'affaccio per la proprietà (1), allora violiamo la proprietà (2).

(2 \Rightarrow 1): Disponiamo di una sequenza di bit b_0, \dots, b_n generata da f e una sequenza di bit r_0, \dots, r_n generata in maniera casuale e indipendente da f , supponiamo che D sia un distinguisher per b_0, \dots, b_n e r_0, \dots, r_n .

Sappiamo che:

$$S_b = b^0 \dots b^{i-1} b^i b^{i+1} \dots b^n$$

$$S_r = r^0 \dots r^{i-1} r^i r^{i+1} \dots r^n$$

Con la tecnica di interpolazione, supponendo che le sequenze differiscono di un solo bit, siamo in grado di affermare che:

$$|P_k^{D,G} - P_k^{D,R}| \geq \frac{1}{2} - k^{-c}$$

$$\left| \sum_{i=0}^{\ell} (P_i - P_{i-1}) \right| \leq \sum_{i=0}^{\ell} |(P_i - P_{i-1})|$$

Quindi

$$\exists i \in \{0, \dots, l\} \quad |P_i - P_{i-1}| \geq \frac{k^c}{2} = k^c$$

$$S_i = b_1 \dots b^{i-1} b^i r^{i+1} \dots r^l$$

$$S_{i+1} = b_1 \dots b^{i-1} b^i r^{i+1} \dots r^l$$

In qualche modo il distinguisher si comporta diversamente quando le sequenze differiscono di un solo bit.

Senza perdita di generalità: $P_{i+1} - P_i \geq k^c$, supponiamo che la probabilità che D restituisce 0 su input $b_1, \dots, b_i, \overline{b_{i+1}}, \dots, r_l$, sia x , allora la probabilità che A indovini il bit successivo è:

$$P[A(b_1, \dots, b_i) = b_{i+1}] = P[r_{i+1} = b_{i+1}] \cdot P_{i+1} + P[r_{i+1} = \overline{b_{i+1}}] \cdot x \\ = \frac{1}{2} \cdot P_{i+1} + \frac{1}{2} \cdot x$$

Possiamo strettamente a cercare una relazione linearmente indipendente con P_{i+1} . Il valore equivalente alla probabilità che il distinguisher restituisca 1 su input $b_1, \dots, b_i, r_{i+1}, \dots, r_l$

$$P_i = P[\text{il bit } i\text{-esimo} = b_{i+1}] \cdot P[D \text{ dia } 1 \text{ su } b_1, \dots, b_i, b_{i+1}, \dots, r_l] + P[\text{il bit } i\text{-esimo} = \overline{b_{i+1}}] \cdot P[D \text{ dia } 0 \text{ su } b_1, \dots, b_i, b_{i+1}, \dots, r_l] = \\ = \frac{1}{2} \cdot P_{i+1} + \frac{1}{2} \cdot (1-x) = \frac{1}{2} \cdot P_{i+1} + \frac{1}{2} - \frac{1}{2} \cdot x$$

Ricaviamo quindi che:

$$x = P_{i+1} + 1 - 2 \cdot P_i$$

Sostituendo x nell'equazione precedente

$$= \frac{1}{2} \cdot P_{i+1} + \frac{1}{2} \cdot P_{i+1} + \frac{1}{2} - P_i \\ = \frac{1}{2} + P_{i+1} - P_i$$

Ma sappiamo che $P_{i+1} - P_i \geq k^c$, quindi:

$$\frac{1}{2} + P_{i+1} - P_i \geq \frac{1}{2} + k^c$$

Blum Blum Shub

È un algoritmo di generazione dei numeri pseudocasuali basato sulla difficoltà di calcolo delle radici quadrate in \mathbb{Z}_n^* .

La funzione di elementi al quadrato è una funzione one-way trapdoor in \mathbb{Z}_{pq}^* , poiché la conoscenza di p e q, ovvero la fattorizzazione di n, permette di calcolare la radice quadrata in tempo polinomiale. Sappiamo inoltre che ogni funzione one-way ammette un hardcore predicate, infatti:

$$\text{LSB}(x) = \overline{fx} \quad \text{un hardcore predicate per l'elemento al quadrato e il bit meno significativo}$$

Il problema è che x ha più di una radice quadrata, che radice bisogna utilizzare?

Primi di Blum

Se p e q sono primi congrui a 3 mod 4 (ovvero $p, q \equiv 3 \pmod{4}$) allora per ogni quadrato x esiste una sola radice quadrata y tale che $y \equiv \sqrt{x} \pmod{n}$.
Sia $n = pq$ con p e q primi di Blum scelti casualmente, sia $x \in \mathbb{Z}_n^*$, allora:

$$\begin{aligned} \mathbb{Z}_n^{n=pq} &= x^2 \cdot (x^2)^* \cdot ((x^2)^*)^* \cdot \dots \cdot (x^2)^l \\ a_1 &\quad a_2 \quad a_3 \quad \dots \quad a_x \quad \left\{ \begin{array}{l} x_1 = x^2 \\ x_{i+1} = x_i^2 \end{array} \right. \\ \text{HCP}(a_2) &= \text{HCP}(a_3), \dots \\ \text{LSB}(a_1) &= \text{LSB}(a_2), \dots \end{aligned}$$

Definiamo quindi $b_i = \text{LSB}(a_i)$ quindi:

$$\begin{array}{ccccccc} a_1 & a_2 & a_3 & \dots & a_x & & \\ x^2 & (x^2)^* & ((x^2)^*)^* & \dots & (x^2)^l & & \\ b_1 & b_2 & b_3 & \dots & b_x & & \end{array}$$

Dove il valore di ritorno sarà la sequenza b_2, b_2, \dots, b_x . Possiamo curiosamente restituire i bit anche in enumerazione crescente, ovvero $b_1, b_2, b_3, \dots, b_x$.

La differenza è che se n è noto e tutti allora è possibile restituire i bit solo nell'ordine b_1, b_2, \dots, b_x .

Blum Blum Shor

$p, q \equiv 3 \pmod{4}$

Sia $D \in \text{PPT}$ un distinguisher

$$P_k^{DU} = P_k[D \text{ rest } 1 \text{ se input è bit casuale}]$$

$$P_k^{DG} = P_k[D \text{ rest } 1 \text{ se input è bit generato}]$$

$$P_k[\{0,1\}^k; D(x) == s]$$

$$P_k[\{0,1\}^k; G(\cdot)]$$

$$\forall D \in \text{PPT} \quad |P_k^{DU} - P_k^{DG}| < k^{-c}$$

Lezione 13

$$|P_k[\{0,1\}^k; b_1, \dots, b_{i-1}, b_i, b_{i+1} = G(s); b = A(b_1, \dots, b_i); b = b_{i+1}] - \frac{1}{2}| < k^{-c}$$

$$|P_k^{DU} - P_k^{DG}| < k^{-c}$$

$$P_k[G_1, \dots, G_i; R_{i+1}, R_{i+2}, \dots, R_{i+k}]$$

$$l \text{ bit } S(i) : G_1 G_2 \dots G_i R_{i+1} \dots R_l$$

$S(0)$: TUTTI CASUALI

$S(l)$: TUTTI GENERATI

$$P(i) = P_k^{S(i)}$$

$$|P(l) - P(0)| > k^{-c}$$

$$\exists i : |P(i) - P(i+1)| > k^{-c'}$$

$$D(b_1, \dots, b_i, R_{i+1}, \dots, R_l)$$

$$A(b_1, \dots, b_i) = \begin{cases} R_i & \text{se } D=1 \\ \bar{R}_i & \text{se } D=0 \end{cases}$$

Sia b_{i+1} il bit generato in posizione i

$$P_k[A(b_1, \dots, b_i) = b_{i+1}] = P_k[R_{i+1} = b_{i+1}] \quad P_{i+1} \\ + P_k[\bar{R}_{i+1} = b_{i+1}]$$

Sia x $P_k[D=1 \text{ se input } b_1, \dots, b_i, \bar{b}_{i+1}, R_{i+2}, \dots]$

$$P_k[A(b_1, \dots, b_i) = b_{i+1}] = \frac{1}{2} P_{i+1} + \frac{1}{2} - \frac{1}{2}x \\ = \frac{1}{2} - \frac{1}{2}(P_{i+1} - x) \quad \rightarrow \frac{1}{2} + \frac{1}{2}(P_{i+1} - 2P_i + P_{i+2}) = \frac{1}{2} + (P_{i+2} - P_i)$$

$$P_i = P_k[R_{i+1} = b_{i+1}] \cdot P_{i+1} = \frac{1}{2} P_{i+1} + \frac{1}{2}x \\ + P_k[\bar{R}_{i+1} = b_{i+1}] \times$$

$$X = 2P_i - P_{i+1}$$

Blum Goldwasser

$n = pq$ p, q primi di blum

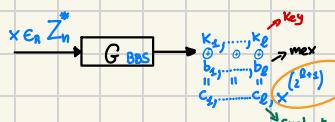
$E(m)$ $m = b_1 \dots b_n$
Encryption

$\begin{matrix} p \\ n \\ s \\ p, q \end{matrix}$

Si vuole creare un crittosistema formato sempre dagli algoritmi di: generazione della chiave, quello di codifica e di decodifica. La chiave è formata dai primi di Blum p, q forniti da K bit, $n = pq$, la chiave pubblica è n , la privata corrisponde alla fattorizzazione di n , ovvero p, q . (problema difficile). Chi possiede la chiave segreta è in grado di calcolare radici quadrate in modulo n .

L'algoritmo di codifica prende che da $x \in \mathbb{Z}_n^*$ e attraverso blum blum shub si generano k_1, \dots, k_l bit casuali. Utilizzati per cifrare un messaggio $m = b_1 \dots b_n$

key₁ ... key_l



primo elemento
della sequenza
generata da blum
blum shub

In sostanza il messaggio viene cifrato con one-time pad, con una chiave pseudocasuale generata mediante un seme
non realmente casuale.

Se k_1, \dots, k_l casuale allora il ciphertext non contiene alcuna informazione sul plaintext, ciò implica che il crittosistema è sicuro

Se si potesse ricevere informazione dal ciphertext riguardo il plaintext, vuol dire che esiste un distinguiser in grado di distinguere qualsiasi coppia di messaggi. Sappendo che ciò è impossibile avendo ora chiave casuale la quale sia valso non può essere distinto da una sequenza casuale; ciò implica l'assurdo.

decodifica

D(c) non fa altro che ricevere l'hard code prediletto, ovvero x^2 ; ciò è possibile sapendo la fattorizzazione di n. Avendo ricevuto $x^{2l}, x^{2l-1}, \dots, x^2$ si calcola k_2, \dots, k_l e rifacendo lo xor con il ciphertext si ottiene il plaintext.

$$m_i = c_i \oplus b_i$$

La complessità del calcolo è $O(l \cdot n^2)$

\downarrow
calcolo di 1
quadrato

Autenticazione - Funzioni (pseudo)-casuali

Necessaria per poter essere certi che quando riceviamo un messaggio il mittente sia quello desiderato e che il messaggio non sia stato modificato.

Metodi per ottenere questa caratteristica:

1. Aggiungere bit di parità
2. Usare codice
3. Usare sistemi non malevoli

Se ho plaintext e ciphertext, però, sono comunque in grado di fare attacchi di plaintext che mi permettono di modificare un messaggio cifrato alterando volontariamente i bit, compresi quelli di autenticazione.

È opportuno addurre sistemi e protocolli di autenticazione affidabili.

Cioè può essere fatto attraverso funzioni casuali che generano una sequenza di bit che permette l'autenticazione.

Le precedenti sono concordate.

Una volta ricevuto il messaggio è possibile calcolare il valore attraverso la stessa funzione calcolata, se i valori corrispondono il messaggio non risulta modificato e l'autenticazione risulta avvenuta.

Dato che sob mittente e destinatario conoscono la funzione, ciò impedisce agli attaccanti di indovinare il codice di autenticazione.

Cioè permette appunto di autenticare gli interlocutori.

La generazione delle funzioni pseudocasuali segue la generazione di bit pseudocasuali, essendo queste generate da un seme (indice).

$$U_k = \left\{ \{0,1\}^k \rightarrow \{0,1\}^k \right\} = (2^k)^{(2^k)}$$

$m, f(m)$

$|A|$

$|A| \cdot |A| \cdot |A| \dots$

↓

$z \rightarrow a_3 \in A$
 $y \rightarrow a_2 \in A$
 $x \rightarrow a_1 \in A$

$|A|^n$

$|A|$ possibilità

$|A|^k$ possibilità

$|A|^k = \left\{ \{0,1\}^k \right\}$

$n = \lg_2 \left((2^k)^{(2^k)} \right) = 2^k \lg_2 2^k = k 2^k$

Gli indici saranno dunque composti da un numero di bit n.

$$\left| \{0,1\}^k \rightarrow \{0,1\}^k \right| = (2^k)$$

$$n = \lg_2 \left((2^k)^{(2^k)} \right) = 2^k \lg_2 2^k = k 2^k \rightarrow \text{sene una quantità esponenziale di bit} \rightarrow \text{NON memorabile per i costi.}$$

Dobbiamo dunque lavorare su un insieme più piccolo.

Definiamo $F_k \subseteq U_k$, sarebbe interessante far in modo che nessuno si accorga che l'insieme considerato sia più piccolo.

1. $|F_k| = 2^k$ definendo così la cardinalità di F_k siamo in grado di generare funzioni pseudocasuali.

2. Esiste algoritmo PPT che calcola

$$(i, x) \mapsto f_i(x)$$

3. Se D è PPT, un algoritmo che, su input 2^k (security parameter di dimensione k) che interroga f_i liberamente e additivamente. Definiamo $P_k^{D,U}$ come la probabilità che D restituisca 1 quando $f_{i,j} \in U_k$ e P_k^{D,F_k} la probabilità che D restituisca 1 quando $f_{i,j} \in F_k$ allo stesso tempo $|P_k^{D,U} - P_k^{D,F_k}| \leq k^{-c/k}$.

$$U_K : \{0,1\}^K \rightarrow \{0,1\}^K$$

$$F_K \subseteq U_K \quad |F_K| = 2^K$$

$$\text{• La funzione } (i, x) \mapsto f_i(x) \in \text{PPT}$$

- Sia D un algoritmo che interrogà f quando vuole

Sia $P_K^{D,U}$ Prob D rest 1 se $f \in F_K$

Sia $P_K^{D,F}$ Prob D rest 1 se $f \in F_K$

$$\left| P_K^{D,U} - P_K^{D,F} \right| < \epsilon^{-w(k)}$$

Non esiste dunque qualcuno in grado di vedere la differenza (POLINOMIALMENTE)

- ① Sia f un pseudo random sequence generator (PRSG), ovvero un algoritmo che genera una sequenza di bit pseudocasuali G, S



Se $x_1 = x_2$ restituiamo lo stesso output, se sto usando le stesse funzioni ovvero se le interrogazioni sono avvenute nello stesso ordine

$$f(x_1)$$

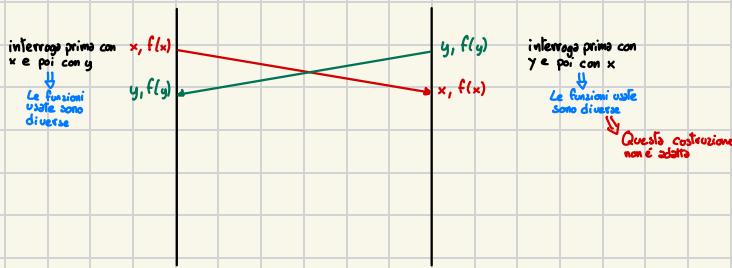
$$f(x_2)$$

$$f(x_3)$$

$$f(x_4)$$

La funzione non è determinata da S ma da S e dall'ordine di interrogazione

Uno dei possibili usi delle funzioni casuali è l'autenticazione dei messaggi → funziona correttamente se lo scambio avviene in maniera sincronizzata



- ② $f_s(x) = x$ - esimo gruppo di K bit generato da $G(s)$, per ogni s c'è una funzione definita in maniera univoca.

Ponendo la condizione che indica l'esistenza di un PPT tale che $(s, x) \mapsto f_s(x)$. Sia $x = 1, \dots, 2^{k-1}$ per come è definito il generatore dobbiamo generare tutti i gruppi precedenti

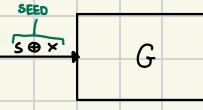
$$f_s(\underbrace{}_K) \quad \text{sequenza di } K \text{ bit ovvero } 2^{k-1}$$

Non riusciamo nemmeno a calcolarlo in tempo polinomiale

$$K(2^{k-1})$$

numero di bit da generare

③ G, S



num
K
primi K bit

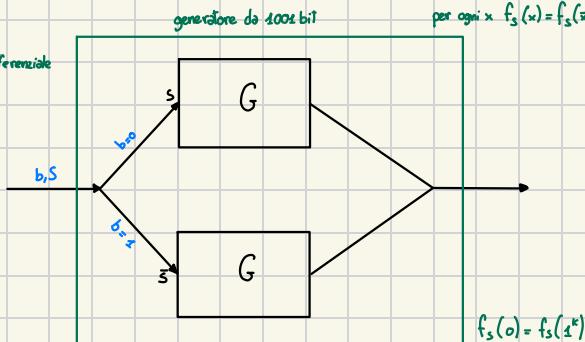
$f_s(x)$ è definito come i primi K bit generati da $G(s \oplus x)$. Sappiamo che ogni indice s determina una funzione univocamente. Inoltre l'algoritmo che genera f_s è polinomiale perché è sufficiente eseguire lo XOR fra s e x e utilizzando il risultato come sema generante i primi K bit.

È necessario ora capire l'eventuale esistenza di un distinguisher in grado di distinguere f da una funzione casuale. Supponiamo che esista D per cui $|P_K^{D(i)} - P_K^{DF}| > k^{-\alpha(n)}$, vorremmo trasformare D in un algoritmo che viola la correttezza di G .

Dobbiamo perciò assicurarsi che il distinguisher stia facendo su G tutti gli esperimenti che sono compatibili con quello che abbiamo detto per definire la correttezza di G .

Il distinguisher D puo' scegliere valori di simboli di x , ma il generatore G conosce tali valori x_1, \dots, x_n prendendo lo XOR di uno dei due valori in input a G , ottenendo lo XOR del sema usato in G , allora il distinguisher è in grado di conoscere l'output di G su due semi di cui è nota la differenza.

$f_s(x)$
 $f_s(y)$



generatore da 1000 bit

per ogni x : $f_s(x) = f_s(y)$

Se interroghiamo G utilizzando $f(x)$ e $f(x')$ sappiamo che i semi utilizzati sono rispettivamente i complementi uno dell'altro. Sappiamo che se $f(x) = a$ e $f(x') = a'$, allora sappiamo che lo XOR dei semi usati per generare ' a ' e ' a' ' è esattamente $x \oplus x'$.

Gli esperimenti che il distinguisher fa su G sono esperimenti che il distinguisher che abbiamo usato per definire la correttezza su G non è in grado di fare.

Il distinguisher utilizzato per le funzioni pseudocasuali è dunque più potente del distinguisher ammesso nella definizione di correttezza di G , e ciò è dunque una contraddizione.

Abbiamo due possibilità:

• È stato definito un generatore troppo debole, è necessaria una definizione più forte che ammetta questo tipo di esperimenti.

• Possiamo far vedere che se abbiamo un generatore di bit pseudocasuali, allora tale generatore resiste ad attacchi di questo tipo

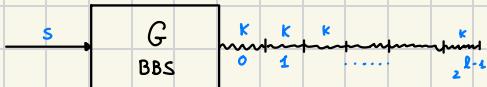
In questo caso è possibile costruire un generatore di bit pseudocasuali G che se usato in questo contesto permette di costruire un distinguisher D . Usiamo il generatore G in uno più grande G' che prende in input dei bit s e lancia una moneta b a caso in input s se $b=0$ e \bar{s} se $b=1$. Selegendo un sema a caso usiamo G con un sema a caso, poiché il complemento di una sequenza casuale di bit resta casuale stiamo generando una sequenza di bit casuali con security parameter $K-s$.

Il nuovo generatore G' è un generatore di bit pseudocasuali che soddisfa la definizione di correttezza. L'output di G' è effettivamente un sema distribuito uniformemente fra tutti i possibili semi, e quindi si comporta come G . Tuttavia usando tale oggetto nel contesto riportato sopra, scopriamo che $f_s(a) = f_s(x')$ e se questo è vero allora D restituiscerebbe s , altrimenti D di fronte ad una funzione pseudocasuale risponderebbe sempre s , di fronte ad una funzione casuale risponderebbe solamente quando $f_s(a) = f_s(x')$, ma tale ipotesi avviene con probabilità 2^{-k} poiché le probabilità che due sequenze di K bit siano uguali, la differenza essendo enorme distingue con probabilità 2^{-k} .

Cerchiamo una soluzione che provi ad evitare il problema

④

Questo metodo si propone come soluzione alle problematiche del secondo metodo che non risolveva in grado di calcolare polinomialmente. Per poter calcolare $f_s(x)$ in tempo polinomiale. Per farlo usiamo un generatore di Blum Blum Shub, dove utilizzando il logaritmo discreto non siamo in grado di calcolare elementi molto distanti, ma se dobbiamo calcolare $f_s(z^{k-s})$ servono solo i bit da $\ell(z^{k-s}-1) \geq \ell^2 \cdot 2^k$



Averendo il sema s per calcolare i bit $K(z^{k-s}-1)$ dobbiamo calcolare $s^{k(z^{k-s}-1)}$, che è complesso da calcolare. Tuttavia gli esponenti lavorano in modulo $q(n)$ e una volta calcolato l'esponente possiamo lavorare i moduli n.

Possiamo calcolare $s^{(z^{k-s}-1) \bmod q(n)}$ in tempo polinomiale l^3 dove l è il numero di bit. Di conseguenza l'oggetto è calcolabile in tempo polinomiale, quindi abbiamo un algoritmo PPT che è in grado di calcolare $f_s(x)$.

Questo è possibile solo grazie al generatore di Blum Blum Shub poiché lavora in un gruppo di cardinalità addito.

$$f_s(x) = x - \text{esima seq. } s^{q(n)} = 1$$

$$y \quad Kx$$

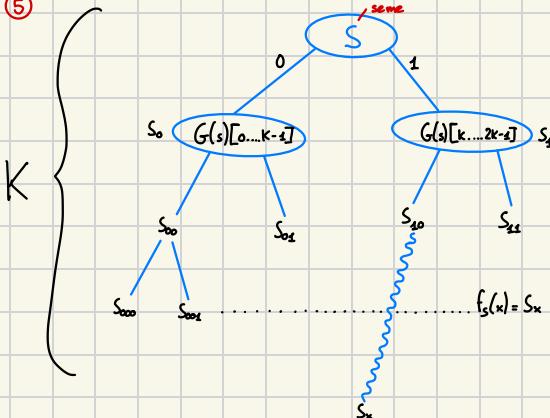
$$s^{(z^{k-s}) \bmod q(n)} = s^{z^{k-s} \cdot q(n)}$$

distinguishers

È necessario dimostrare l'indistinguibilità del generatore da una funzione realmente casuale.

D interroga la funzione f , facendo in realtà esperimenti su G , i quali possono avvenire in tempo polinomiale. Non vi è dunque la possibilità con il distinguisher di vedere una quantità di bit esponenziale come avviene il generatore. Non riusciamo però a costruire un controesempio.

⑤



Usiamo l'indice della funzione s di k bit come seme utilizziamo un generatore pseudocasuale $G(s)[0, \dots, k-1]$ ouero s e lo stesso per $G(s)[k, \dots, 2k-1]$ ouero S_1 . E ripetiamo ricorsivamente il procedimento.

Scendendo nell'albero, arriviamo al seme indicato da x , ouero S_x , che è un seme di k bit, che è il risultato della funzione $f_s(x)$. Ovviamente questo albero ha 2^k nodi e le foglie rappresentano i valori delle funzioni sui diversi argomenti a partire da s in radice, quindi la funzione $f_s(x)$ ouero $f_s(x)$ è ben definita.

Calcolando solo un cammino dell'albero il tempo di calcolo risulta polinomiale.

Infatti per calcolare un nodo dell'albero nel caso peggiore calcoliamo $2k$ bit per K volte a cui si moltiplica il tempo di generazione di $2k$ bit del generatore di Blum Blum Shub che esegue un elevamento al quadrato per ogni bit, quindi il tempo di calcolo è $\Theta(k^4)$.

Dobbiamo verificare che non esiste un distinguisher D che sia in grado di distinguere la funzione $f_s(x)$ da una funzione realmente casuale. Creiamo un distinguisher D che interroga la funzione $f_s(x)$ implicitamente sba facendo degli esperimenti sul generatore G in maniera nidiata.

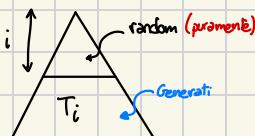
Guardando l'albero a livelli un generatore pseudocasuale trasforma una certa quantità di casualità in una quantità di casualità diversa.

Definiamo gli alberi:

$$T_0 = s^t \dots s^{i-1} s^i s^{i+1} s^{i+2} \dots s^k$$

$$T_i = r^t \dots r^{i-1} r^i r^{i+1} r^{i+2} \dots r^k$$

Dove T_0 è l'albero che rappresenta esattamente l'albero definito per la generazione di funzioni pseudocasuali, mentre T_i corrisponde ad una funzione casuale campionata uniformemente dall'insieme U_k .



Definiamo quest'albero (non possiamo creare perché ha dimensione esponenziale)

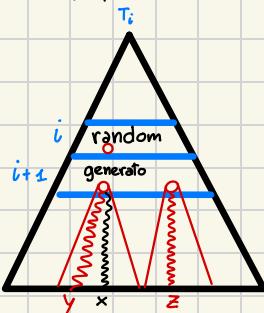
Il seme è random, il resto è generato

T_0 è la funzione pseudocasuale

T_k è la funzione casuale

$$P_i \triangleq P_k^{D, T_i} - P_k^{D, U} > k^{-c}$$

$$\exists i \quad |P_i - P_{i+1}| > k^{-c}$$



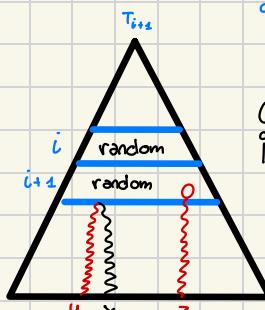
Il distinguisher secondo la nostra ipotesi vede la differenza tra un albero costruito secondo la prima tecnica e un albero costruito secondo la seconda.

$$|P_k^{D, G} - P_k^{D, U}| > k^{-c} \quad \text{quindi} \quad |P(i) - P(i+1)| > k^{-c}$$

In questo caso costruiamo tutti gli alberi intermedi due per ogni livello $i=0, \dots, k$, definiamo l'albero T_i tale che:

- I nodi dei livelli $0, \dots, i$ sono casuali

- I nodi dei livelli $i+1, \dots, k$ sono pseudocasuali



Con la stessa tecnica di interpolazione possiamo affermare che se il distinguisher è in grado di ricevere la differenza tra un albero costruito secondo lo schema T_0 e un albero costruito secondo lo schema T_k , allora può individuare la differenza tra i livelli addizionali mediante interpolazione.

Poiché due alberi addiacenti differiscono solamente per un livello specifico, otteniamo due alberi distinti che si differenziano solo per quel livello particolare.

Perfatto, calcolando $f_s(x)$ risulterebbe equivalente a retrocedere al nodo i -esimo dell'albero T e osservare casualmente in un caso e pseudocasualmente nell'altro.

CON UN SEME GENERATO AL LIVELLO PRECEDENTE IN MODOVERAMENTE CASUALE. L'esperimento di calcolare $f_s(x)$ è quindi il classico test su un generatore di bit pseudocasuali.

Sto facendo più esperimenti indipendenti

$$\exists i \in \{0, \dots, k-1\} \quad |P(i) - P(i+1)| > k^{-c}$$

Effettuando un esperimento su $f_s(x)$ che proviene dall'istanza dello stesso nodo i -esimo dell'albero T , allora può essere considerato un nuovo esperimento indipendente, consentendo la realizzazione di più esperimenti indipendenti. Se la possibilità di successo con un singolo esperimento è inferiore a qualsiasi polinomio.

Di conseguenza l'esistenza di un attaccante al generatore di funzioni si traduce in un attaccante al generatore di bit pseudocasuali G , quindi il distinguisher non esiste se assumiamo che il generatore di bit sia soddisfacente.

Il sistema così costruito è dimostrabilmente sicuro; viene costruita una funzione $f: \{0,1\}^k \rightarrow \{0,1\}^k$, partendo da una funzione pseudocasuale di k bit in K bit, che incrementa la dimensione dell'input. Possiamo concepire questo concetto come un'estensione dell'idea di generazione di funzioni pseudocasuali. Supponiamo di avere una funzione scelta in modo uniforme, tra le funzioni da $\{0,1\}^k$ a $\{0,1\}^k$, oppure una funzione da K bit a K bit, selezionata da un insieme di un insieme di dimensione più ridotta, poiché dipende dalla stessa seme utilizzata per f . Il nostro obiettivo è affermare che questo sistema è sicuro nel senso che non esiste alcun algoritmo probabilistico polinomiale in grado di distinguere una funzione pseudocasuale da una funzione casuale.

L'esempio forger può essere convertito in un distinguisher, pertanto, affermare che si tratta di un sistema di autenticazione o di una funzione pseudocasuale da K bit a K bit, e dimostriamo che è una funzione pseudocasuale nel senso delle definizioni date sulle funzioni pseudocasuali, ovvero non esiste alcun algoritmo probabilistico polinomiale in grado di distinguere tra funzione pseudocasuale da una funzione casuale.

Per farlo costruiamo $F_i(m_1, \dots, m_l)$: prendere i primi i blocchi del messaggio e applica la funzione f pseudocasuale a questi blocchi e per i successivi $l-i$ blocchi applica la funzione r casuale, quindi:

$$F(m_1, \dots, m_l) = r(m_2 \oplus \dots \oplus r(m_{i+1} \oplus f(m_1 \oplus \dots \oplus f(m_3 \oplus f(m_2 \oplus f(m_1)))))$$

A questo punto sapendo che f_0 è sempre casuale in tutti i punti e f_1 è pseudocasuale in tutti i punti, sapendo distinguere i due estremi, possiamo, per interpolazione, distinguere anche i punti intermedi. Supponiamo che $P(i)$ è la probabilità che il distinguisher restituisca i se la funzione di autenticazione è la funzione F ; allora:

$$|P(i) - P(i+1)| > k^{-c}$$

Quindi distinguere sul livello i -esimo e il livello $l-i$ -esimo si traduce in una distinzione nell'utilizzo di una funzione casuale, quindi:

$$\exists i \in \{0, \dots, k-1\} \quad |P(i) - P(i+1)| > k^{-c'}$$

Diventando quindi un distinguisher per la funzione pseudocasuale f .

Problema della non ripetibilità

La necessità è quella di avere un sistema che permetta di autenticare un messaggio, in modo che il mittente non possa negare di averlo inviato. Con le tecniche finora studiate infatti chiunque potrebbe creare i messaggi.

Siamo parlando del concetto di firma digitale, chi firma il messaggio non può negare di averlo fatto, in quanto la firma è univoca e non può essere riprodotta da nessun altro.

Vorremmo che solo un'agente possa firmare il messaggio e che tutti possano verificare la firma.

Devono quindi esserci due chiavi, una privata per firmare e una pubblica per verificare.

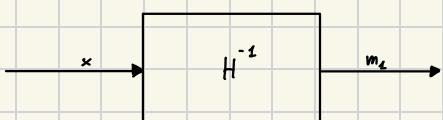
Dalla chiave pubblica non devo poter rivelare alla privata

Una funzione hash H è one-way se dato x è difficile trovare un qualsiasi m tale che $H(m) = x$, questo se è facile da calcolare ma difficile da invertire. In una funzione iniettiva, dove ogni elemento dell'immagine (codominio) ha un unico elemento dell'insieme di pertinenza, quindi trovare l'inversa è semplice, siccome $f: A \rightarrow B$ è iniettiva allora $f^{-1}: B \rightarrow A$ è definita. Per invertire una funzione hash, che non è iniettiva, è necessario utilizzare una tecnica di **brute force** ovvero provare tutti i possibili input e trovare un qualsiasi input che produca l'hash x .

Se H è una funzione collision resistant, allora H è anche una funzione one-way.

Dim Supponiamo che H non sia one-way, mostriamo quindi che H non è collision resistant.

Sia m un messaggio casuale e calcoliamo $x = H(m)$, quindi x è un valore casuale che viene dato in input alla macchina che inverte la funzione hash e che produrrà un messaggio m' tale che $H(m') = x$.



Con il fatto di aver scelto m in modo casuale, la probabilità che H^{-1} restituisca lo stesso messaggio m è trascurabile, quindi con probabilità 1 abbiamo trovato una collisione.

Dim alternativa

Supponiamo che H non sia one-way.

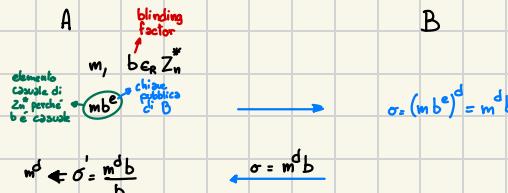


Se costruisco un algoritmo A in grado di invertire una funzione hash, così facendo trovo con probabilità elevata una collisione. Ma la funzione hash è collision resistance. Assurdo. Quindi H è one-way.

Blind Signature

La firma va apposta su un messaggio dove il firmatario non conosce cosa sta firmando

B deve firmare un messaggio m mandagli da A senza conoscerne m



Ho (m, o') essendo $o' = m^d$ questo messaggio firmato con la chiave pubblica di B.

Con le opportune sanzioni questo sistema va più che bene, per avere la garanzia per B che il messaggio sia costruito in maniera adeguata a quello che dovo effettivamente firmare. Per farlo abbiamo due possibilità:

1. La chiave pubblica usata da B può essere usata solo per spedire pin.

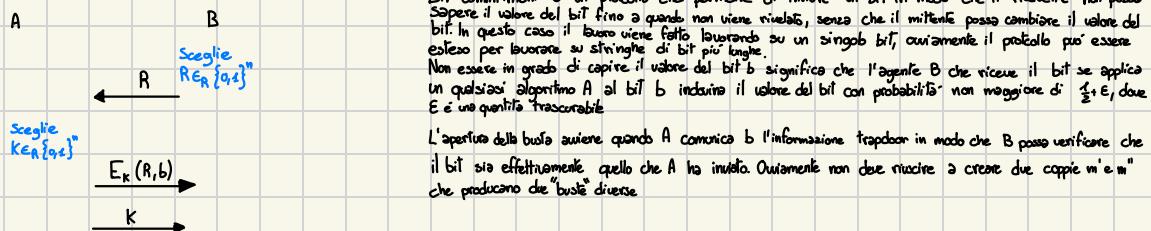
2. Complicando il protocollo. Per farlo A genera m_1, m_2, \dots, m_d differenti messaggi di comunicazione pin e tali messaggi devono essere tutti identici, differendo solo per il valore del pin. A opera $b_{1, \dots, d}$ blindin factor secondo il protocollo e spedisce a B i messaggi $m_1 \cdot b_1, m_2 \cdot b_2, \dots, m_d \cdot b_d$. A questo punto B sceglie $i \in \{1, \dots, d\}$, scegliendo quindi una lettera e comunica b ad A, a questo punto A comunica $m_1, m_2, \dots, m_{i-1}, b_{i-1}, m_{i+1}, b_{i+1}, \dots, m_d, b_d$ a B, comunicando tutte le informazioni tranne quelle relative al messaggio i-esimo. B verifica la correttezza dei dati ricevuti e se la verifica ha successo produce la firma $o' = (m_i \cdot b_i)^d$; altrimenti rifiuta la firma.

Scegliendo l'attacco reale, lo sufficientemente grande, la probabilità che B riesca a trovare il valore del pin si abbassa, ma la proprietà di attacco resta comunque alta, implicando quindi la necessità di trovare altri strumenti per contrastare gli attacchi.

Lezione 11

Bit commitment

Primo protocollo

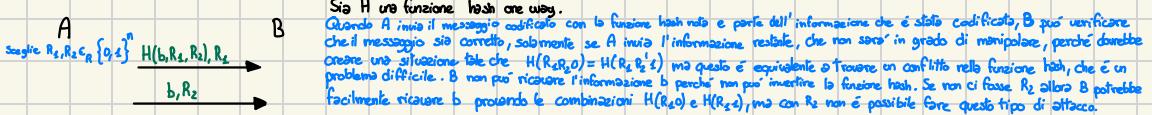


Sulla base delle analisi fatte fino ad ora, non è possibile ricevere la chiave di cifratura K a partire dal plaintext, anche se sceglio parte del plaintext. B verifica l'autenticità del messaggio ricevuto non appena riceve la chiave K, ovvero l'informazione trapdoor. A non può imbrogliare perché trovere due chiavi K_0 e K_1 tali per cui $E_{K_0}(0, n) = E_{K_1}(1, n)$, ma su un algoritmo di crittografia simmetrica come AES non è possibile trovare due chiavi che producano lo stesso ciphertext.

In tale algoritmo ci sono due problemi:

1. B deve collaborare, altrimenti non può fare nulla.
2. Non è buona cosa scegliere un protocollo che si basa su un algoritmo di crittografia non dimostrabilmente sicuro.

Secondo protocollo



Terzo protocollo

Questo protocollo è dimostrabilmente sicuro, ma in termini di complessità è molto più costoso degli altri due.

A

B

Sceglie $R \in \{0,1\}^l$
dove $R = r_1 r_2 \dots r_l$

R

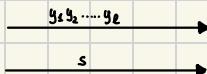
• genera S per PRG G
 $SC_R \{0,1\}^n$

• genera una pseudosequenza
di bit $x_R \{0,1\}^l$

• Date $x = x_1 x_2 \dots x_l$

• Vi: invia a B $\begin{cases} x_i & \text{se } R_i=0 \\ x_i \oplus b & \text{se } R_i=1 \end{cases}$

• Produce $y_1 y_2 \dots y_l$



La sequenza y inviata è una sequenza di bit o in maniera pseudocasuale. O in XOR con il bit b quando $r_i=1$, nel momento in cui un oggetto è casuale, lo è anche la sua XOR con un altro oggetto, non contenendo quindi nessuna informazione circa il bit b .

Una volta che B riceve il semme s , lui è in grado di ricostruire la sequenza x e quindi verificare che tutti gli altri y_i siano corretti; se così non fosse, allora B può interrompere la comunicazione.

A non può imbrogliare perché dovrebbe trovare due semi s_0 e s_1 tali per cui s_0 porta a y_1, \dots, y_l e s_1 porta a y'_1, \dots, y'_l e per ogni i se $r_i=1$ allora $x_i=x'_i$, ma per essere più precisi $0 \otimes x_i = s \otimes \bar{x}_i$

Schema barriera (n, n)

Negli schemi a barriera disponiamo di un segreto s che vogliamo proteggere, lo distribuiamo ad n agenti creando delle parti delle share s_1, \dots, s_n , in modo che mettendo assieme n parti si possa ricostruire il segreto, ma con meno di n parti non si possa ricavare alcuna informazione sul segreto, quindi nominato un'informazione binaria del segreto s .

Non posso costruire una sequenza in n parti, perché altrimenti se si riunissero meno di n agenti otterrebbero comunque delle informazioni.

Schema XOR

$$x_1, x_2, \dots, x_{n-1} \in \{0, 1\}$$

$$x_n = \left(\bigoplus_{i=1}^{n-1} x_i \right) \oplus s$$

solo quando ha tutte le parti riesco a leggere il segreto

$$x_n \oplus \left(\bigoplus_{i=1}^{n-1} x_i \right) = s$$

Avere x_1, \dots, x_{n-1} sequenza di bit casuali, e essendo il segreto s , sia $s_n \equiv s \oplus s_1 \oplus \dots \oplus s_{n-1}$ essendo s_1, \dots, s_{n-1} casuali, s_n è casuale, quindi non contiene alcuna informazione sul segreto s , e lo sarebbe di qualunque sottinsieme. Vale infatti che, per come è stato costruito $s_k \oplus s_n = s$, e lo XOR con qualunque sottinsieme di $s_1, \dots, s_k = S \oplus s_{k+1} \oplus \dots \oplus s_n$, non fornendo quindi alcuna informazione sul segreto s , e lo sarebbe di qualunque sottinsieme.

Vale infatti che, per come è stato costruito, $s_k \oplus s_n = s$, e lo XOR con qualunque sottinsieme di $s_1, \dots, s_k = S \oplus s_{k+1} \oplus \dots \oplus s_n$, non fornisce alcuna informazione sul segreto.

Questo schema ha però un problema, un agente non collaborativo potrebbe non inviare la sua parte e quindi impedire la ricostruzione del segreto, non vorremo che fosse possibile ricostruire il segreto anche con un numero minore di parti, ma non con meno di un numero fisso k .

Schema di Shamir

Voglio costruire uno schema (n, k)

Se voglio costruire un messaggio che si decifri con K agenti su n

Lo schema di Shamir è un metodo per distribuire un segreto s in n parti in modo che con K parti si possa ricostruire il segreto, ma con meno di K parti non si possa ricavare alcuna informazione sul segreto s .

Teorema

Un polinomio di grado $K-1$ è individuato unicamente da K punti. Per K punti passa esattamente un polinomio di grado $K-1$

Costruiamo un polinomio P casuale di grado K tale che $P(a) = s$ ^{secret}

$$P(x) = a_0 + a_1 x + a_2 x^2 + \dots + a_{K-2} x^{K-2}$$

$$P(0) = a_0 = s$$

Scegliendo quindi: a_1, \dots, a_{K-2} casuali e $a_0 = s$ otteniamo un polinomio P casuale.

Gli agenti sono n , e ad ognuno di essi viene assegnato il valore del polinomio P in un punto casuale x_i , che non sia il punto 0.

Ogni agente non può dedurre nulla circa il segreto complessivo.

Per ricostruire il segreto s , è necessario un insieme di almeno K parti. Questo è garantito dal teorema precedentemente enunciato, secondo il quale K punti sono sufficienti per individuare unicamente un polinomio di grado $K-1$.

La ricostruzione del segreto può essere effettuata utilizzando una tecnica come l'interpolazione polinomiale. Ad esempio, utilizzando l'interpolazione di Lagrange, il segreto può essere recuperato con la seguente formula:

$$s = P(0) = \sum_{i=1}^k P(x_i) \prod_{j=1, j \neq i}^k \frac{x_i}{x_i - x_j}$$

Dove x_1, x_2, \dots, x_k sono i punti in cui gli agenti hanno i valori del polinomio P .

Zero Knowledge

| crittografi mangiatori

Ci sono 3 crittografi che vanno a cena

A un tavolo a fianco c'è un tipo sopratto che li osserva

Arriva il conto ma è già stato pagato

| crittografi vogliono sapere se a pagare è stato uno di loro

L'idea è quella di usare un protocollo di **zero knowledge**, ovvero un protocollo che permette di dimostrare di conoscere una certa informazione senza rivelarla. Ognuno di loro nasconde un bit b_i che indica se ha pagato o meno il conto, il bit verrà 1 se ha pagato e 0 altrimenti. Ovviamente è chiaro che se il risultato di tutti i bit è 0 allora nessuno ha pagato, se invece il risultato è 1 allora ogni crittografo sa che almeno uno di loro ha pagato, ma non sa chi.

Nessuno vuole calcolare

$$f(c_1, c_2, c_3) = \text{XOR}$$

senza sapere nulla sui c_{other} , ma sapendo che i partecipanti sono onesti

Ognuno dei crittografi genera un bit lanciando una moneta, ogni crittografo vede il risultato della propria moneta e del compagno alla propria sinistra. Ogni crittografo calcola b XOR delle due monete a cui ha accesso



A questo punto indiamo che ogni singolo elemento viene considerato due volte, perfetto il risultato dello XOR di tutti i bit sarà 0 se nessuno ha pagato e 1 altrimenti.

Ogni crittografo sa quello che puo' sapere conoscendo il proprio bit e il risultato della funzione, quindi da questo protocollo ogni crittografo puo' apprendere solo ed esclusivamente cio' che si puo' apprendere dalla conoscenza del proprio argomento e dalla conoscenza del risultato della funzione.

Chieramente l'agente che sa di essere il crittografo che ha pagato sa il risultato degli altri crittografi.

Generalizzazione del protocollo

Il protocollo puo' essere generalizzato a n crittografi, ognuno dei quali lancia una moneta e ognuno di loro vede il risultato delle monete dei crittografi alla propria sinistra, calcolando lo XOR tra le due monete. Ogni singola moneta viene conteggiata due volte, quindi il risultato dello XOR di tutte le monete sarà 0 se nessuno ha pagato, 1 altrimenti (essendo il complemento).

Agente esterno che incarica la trasmissione di un messaggio

Supponiamo che qualcuno incarichi un crittografo di spedire un bit, ma non vuole comunicare chi è l'incaricato. Se il crittografo in questione si comporta come l'unico crittografo che puo' aver pagato, ha la capacità di spedire un bit senza che nessuno si accorga di chi è stato l'incaricato.

Agente interno che vuole spedire un messaggio

Supponiamo che il tempo sia scandito da un clock. Al primo ciclo di clock, il crittografo i lancia la moneta e la passa al crittografo alla sua destra. Al secondo ciclo di clock, il crittografo i calcola lo XOR tra la moneta che ha ricevuto e la moneta che ha lanciato. Il crittografo che vuole spedire il bit complementa il risultato chi non vuole spedire il bit lascia il risultato così com'è.

Supponendo che il crittografo che vuole spedire il bit sia uno solo, riuscirà a spedire il bit con successo.

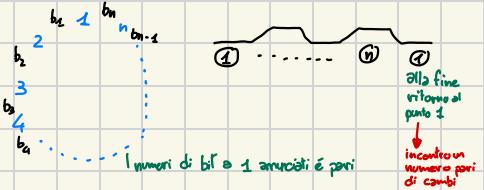
Con questo protocollo però potrebbero esserci dei conflitti, ovvero potrebbe capitare che due crittografi vogliono spedire un bit.

I crittografi hanno modo di accorgersi di questo conflitto, ogni crittografo riesce ad assegnare il bit che è stato spedito nella rete, se il bit è diverso da quello che ha spedito lui, allora sa che c'è stato un conflitto. In questo caso i crittografi che hanno spedito il bit si accorgono che il risultato non è andato a buon fine, spediscono il bit ad un ciclo di clock casuale e riprovano.

Agente interno che vuole spedire un messaggio ad un agente incaricato

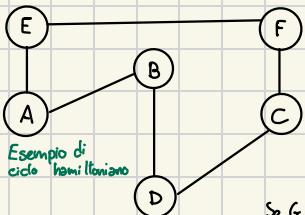
per farlo basta utilizzare il metodo precedente, originando il messaggio con la chiave pubblica del destinatario. In questo modo solo il destinatario potrà leggere il messaggio.

Il risultato è un sistema di comunicazione completamente non tracciabile, non si conoscono: mittente, destinatario e contenuto. Il protocollo ha però dei difetti, e infatti molto dispendioso in termini di risorse, poiché tutti gli agenti devono essere sempre attivi ed è soggetto ad un attacco chiamato **denial of service**, se qualcuno vuole oscurare il canale puo' trasmettere sequenze di bit casuali e nessuno saprà mai chi ha ascoltato il canale.



Ciclo Hamiltoniano

Un ciclo Hamiltoniano è un ciclo che passa per tutti i vertici di un grafo una e una sola volta.



P permette G per ottenere H , grafo isomorfo a G e codifica tutti i bit della matrice di adiacenza di H con bit commitment, e sia H' il risultato. P invia H' , sapendo che il dato inviato non rivelano informazioni circa i bit al suo interno.

V lancia una moneta per decidere se chiedere a P l'evidenza che $G \cong H$ o l'evidenza che H ammette ciclo hamiltoniano. P obbedisce e nel caso in cui V abbia chiesto l'evidenza che $G \cong H$ allora risponde H e l'isomorfismo tra G e H , infatti se uno ammette ciclo hamiltoniano lo ammette anche l'altro. Se invece V ha chiesto l'evidenza che H ammette ciclo hamiltoniano, scopre solo i bit di H che costituiscono un ciclo hamiltoniano.

Se G non ammette ciclo hamiltoniano allora il grafo H inviato al prover, non è possibile che sia, sia isomorfo a G e sia che ammetta un ciclo hamiltoniano. Se è isomorfo a G , visto che G non ammette ciclo hamiltoniano, allora H non può ammetterlo. In quel caso il prover sopra solamente fornire la risposta ad una delle due domande. Visto che il verifier sceglie le domande casisticamente, con probabilità $\frac{1}{2}$ il verifier sceglierà la domanda a cui il prover non sa rispondere. Di conseguenza se G ammette ciclo hamiltoniano il prover risponderà sempre in qualsiasi caso e quindi risponderà correttamente con probabilità 1.

G grafo

P permette G in G'

codifica matrice di adiacenza di G' in H

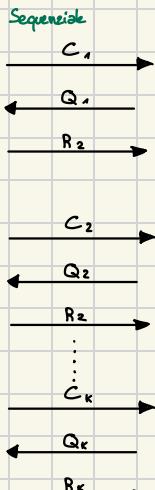


P

Risposta 1 Apre solo le buste di un ciclo

2 Apre tutte le buste e invia una permutazione

Ripetizione parallela della zero knowledge



Per come abbiamo definito la zero Knowledge, il protocollo può essere ripetuto più volte eseguendo arbitrariamente la sequenza di challenge, questione, risposta. Se il protocollo seguisse uno schema parallelo di esecuzione, ovvero se il verifier ponesse più domande in parallelo, il prover potrebbe rispondere a tutte le domande con un singolo risposta, senza dover ripetere il protocollo per ogni domanda.

Per uno schema sequenziale, dove gli eventi sono mutualmente indipendenti, è possibile creare misure di probabilità che risultano indistinguibili dall'originale. Tuttavia, la zero knowledge proof in uno schema parallelo presenta delle sfide. Questo è evidente definendo q_1, q_2, \dots, q_g come una funzione hash unidirezionale $H(c_1, c_2, \dots, c_g)$. Nel contesto dello schema parallelo, la tecnica per simulare la sequenza di messaggi diventa impattabile. La difficoltà principale risiede nella necessità di invertire la funzione hash, che è di tipo unidirezionale, al fine di costruire le challenge c_1, c_2, \dots, c_g per la simulazione.

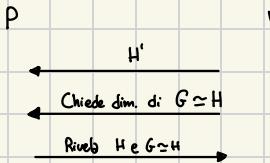
Una proof Knowledge in cui il verifier pone domande al prover sulla base di una funzione hash one way applicata alle challenge è qualcosa che riesce a convincere terza parti che il prover conosce la soluzione.

Non interactive zero Knowledge

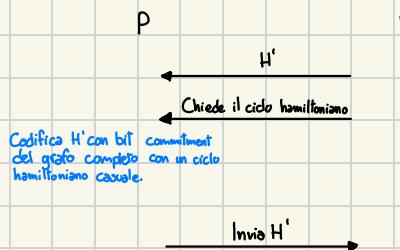
Lo non interactive zero Knowledge è una variazione della zero Knowledge proof in cui qualcuno conosce il segreto senza rivelare chi sia, non rivelando nessuna informazione circa il segreto stesso. L'aspetto cruciale è la capacità di dimostrare la conoscenza di un segreto in modo autonomo e senza rivelare dettagli sensibili, rendendole particolarmente utili in applicazioni come le blockchain.

Costruzione del simulatore H

Vogliamo far sì che V non possa utilizzare l'interazione con P per convincere terzi. Per farlo casualmente la domanda di V, se la domanda che fa V è di dimostrare che $G \cong H$, allora costruiamo H' secondo il protocollo.



Se invece la domanda è di dimostrare che H ammette un ciclo hamiltoniano, allora P invia H' bit commitment del grafo completo, scegliendo una permutazione casuale dei nodi come ciclo.



La distribuzione di probabilità delle domande che può effettuare il prover è di $\frac{1}{2}$ per ciascuna domanda. La distribuzione di probabilità delle risposte che può dare il prover è corretta nel primo caso è infatti: doppia alla permutazione casuale che può assumere il grafo. Nel secondo caso invece si tratta di una permutazione casuale dei nodi del grafo H' . Permettendo un grafo con ciclo hamiltoniano, il ciclo diventa una permutazione dei nodi del grafo, quindi qualunque permutazione ha la stessa probabilità di essere un ciclo hamiltoniano.

Nel primo caso la misura di probabilità è sempre lo stesso, nel secondo caso invece la misura di probabilità è diversa.

Un qualsiasi terzo osservatore che abbia potenza di calcolo polinomiale che avverte l'interazione fra P e V , non può distinguere l'interazione reale che avviene fra P e V da un'interazione simulata fra P e V . Un osservatore che avverte H' fornito dalla seconda interazione (avendo quello che ha la permutazione casuale dei nodi del grafo completo), non riesce a capire che l' H' inviato dal verifier è la matrice di adiacenza di un grafo completo o meno.

Supponendo di avere un algoritmo D e PPT che campiona i messaggi fra View e H e che riesce a distinguere tra l'interazione reale e quella simulata, allora siamo in grado di distinguere tra il bit commitment di un grafo con ciclo hamiltoniano da un bit commitment di un grafo completo, ma ciò si traduce in un algoritmo che riesce il problema del bit commitment, che è un problema non risolvibile in PPT .

Quello che viene effettivamente visto fuori è la distribuzione di probabilità tra le domande e le risposte, ma non viene effettivamente usata la differenza tra le due versioni di H' .

Perfect zero Knowledge $\text{View}(P, V, x, h) = H(P, V, x, h)$ le due distribuzioni di probabilità sono identiche.

Statistical zero Knowledge $\sum_{\alpha} | \text{VIEW}(x, h)[\alpha] - H(x, h)[\alpha] | < k^{-w/4}$ la differenza tra le due distribuzioni è trascurabile, ovvero non esiste un algoritmo PPT in grado di distinguere le due distribuzioni.

Computational Zero Knowledge $\text{VIEW}(x, h), H(x, h)$ POLINOMIALMENTE INDISTINGUOGHIBILI ovvero $\exists D \in PPT$ sia P_K^{DV} la probabilità che D restituisca 1 campionando da view e sia P_K^{DH} la probabilità che D restituisca 1 campionando da H ,

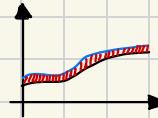
$$| P_K^{DV} - P_K^{DH} | \leq k^{-w/4}$$

Non esiste un algoritmo PPT che riesce a distinguere tra le due distribuzioni di probabilità.

Lezione 19

View (x, h) ^{hint} (α)

esiste algoritmo polinomiale
 $M(x, h)$ (α)



• perfect zero-knowledge $\forall x \forall h \quad \text{View}(x, h) = M(x, h)$

• statistical zero-knowledge $\sum_{\alpha} |\text{View}(x, h)(\alpha) - M(x, h)(\alpha)| < K^{-w(z)}$

• computational zero-knowledge $\text{View}(x, h), M(x, h)$ sono computazionalmente indistinguibili

Grafo tricolorabile

Nel contesto della dimostrazione della tre-colorabilità di un grafo senza rivelare la colorazione effettiva dei nodi, si adotta un protocollo computational zero knowledge. L'obiettivo è dimostrare che un grafo può essere colorato con tre colori in modo tale che nodi adiacenti abbiano colori distinti, senza mai rivelare la specifica colorazione adottata.

Il protocollo inizia con un accordo fra il prover e il verifier su un grafo G . Il prover si impegna a dimostrare la tre-colorabilità del grafo senza rivelare direttamente la colorazione.

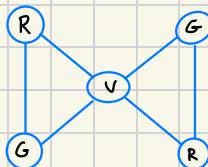
Per mantenere la confidenzialità, il prover sceglie casualmente una permutazione tra i tre colori (ad esempio rosso, verde e blu) senza rivelarla al verifier. Questa permutazione sarà utilizzata per colorare i nodi del grafo.

Il verifier, a sua volta, sfida il prover selezionando casualmente due nodi adiacenti, A e B , e richiede la rivelazione dei colori associati secondo la permutazione scelta. Il prover risponde rivelando i colori senza svelare la permutazione effettiva. Il verifier può verificare se i colori sono diversi, confermando così la corretta tre-colorabilità del grafo. Questo processo può essere ripetuto per diverse coppie di nodi di nodi adiacenti, ma la casualezza nella selezione delle sfide impedisce al verifier di dedurre la permutazione specifica attraverso tentativi ripetuti.

Il prover, per evitare la possibile deduzione della permutazione da parte del verifier attraverso esperimenti multipli, permette casualmente i colori ad ogni sfida. Così facendo il verifier non può accumulare informazioni utili per dedurre la tre-colorabilità del grafo.

Alla conclusione del protocollo, se il prover ha superato con successo tutte le sfide, il verifier acquisisce la convinzione della tre-colorabilità senza mai venire a conoscenza della specifica permutazione dei colori utilizzata dal prover. Questo dimostra l'efficacia della computational zero knowledge nel preservare la riservatezza della colorazione dei nodi.

$$\left(\frac{E-1}{E}\right)^n < \frac{1}{2}$$



Concreto

Alla fine il prover facendo gli esperimenti ho in mano la tri-colorazione \rightarrow non è zero knowledge

Tra un esperimento e l'altro scambio completamente i colori, così il prover non ottiene informazioni in più

ad es $G \rightarrow V$
 $V \rightarrow R$
 $R \rightarrow G$

Io tricoloro il grafo e invio tramite bit commitment

rivelò solo un nodo e i suoi adiacenti, poi permuto

Al posto di uoni in Z_n^* Uso punti su una parabola in Z_n^*

$$y = mx + q$$

y e x sono punti di un gruppo finito (Z_n^*)Gruppo $(A, *)$ $g \in A$ ordine di G: più piccolo t.t.c. $g^t = 1$ $\{g^1, g^2, \dots, g^{t-1}\}$ ha un sottogruppoa trova $x = a \cdot g^x$
Log Discreto

$$g^x \cdot g^y \cdot g^{-z} = g^x$$

$$g + g + g = 3g$$

Diffie-Hellman

A (x, g^x)

B (y, g^y)

Notazione Additiva

(x, xy)

(y, yg)

a trova x $a \cdot g^x$
Log discreto sulla curva ellittica è difficile.

$$(g^y)^x = (g^x)^y$$

sono lo stesso
oggetto

$x(yg) = y(xg)$

$(xy)g$

Crittosistema di El Gamal (notazione multiplicaiva)

(x, g^x)

E(m)

$k \in \{a \text{ ordine}(g) - 1\}$

$C_1 = g^k$

$C_2 = mg^k$

$\text{RET}(c_1, c_2) = (g^x, mg^x)$

$D(c_1, a) \text{ RET } c_2 \cdot c_1^{-x}$

$$mg \cdot (g^k)^{-x} = mg^{x-k} = m$$

Crittosistema di El Gamal (notazione additiva)

(x, g^x)

E(m) $c_1 \leftarrow kg$

$c_2 \leftarrow m + kg$

RET (c_1, c_2)

$D(c_1, a) \text{ RET } c_2 + (-x)c_1$

$m + kg - (xk)g = m + kg - xkg$

$y = xy$

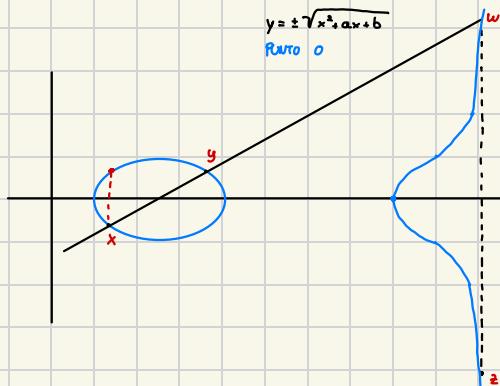
Curva ellittica

$$y^2 + axy + by = x^3 + cx^2 + dx + e$$

con a, b, c, d, e numeri scelti

$$y^2 = x^3 + ax + b$$

Studiamo la curva di questo genere
chiamiamo d a
e chiamiamo c b



Se prendo due punti non simmetrici,
la retta passante per due punti incrocia
la funzione in un terzo punto

Somma di due punti simmetrici è Punto all'infinito.

Somma di tre punti allineati è Punto all'infinito

$$x+y=2$$