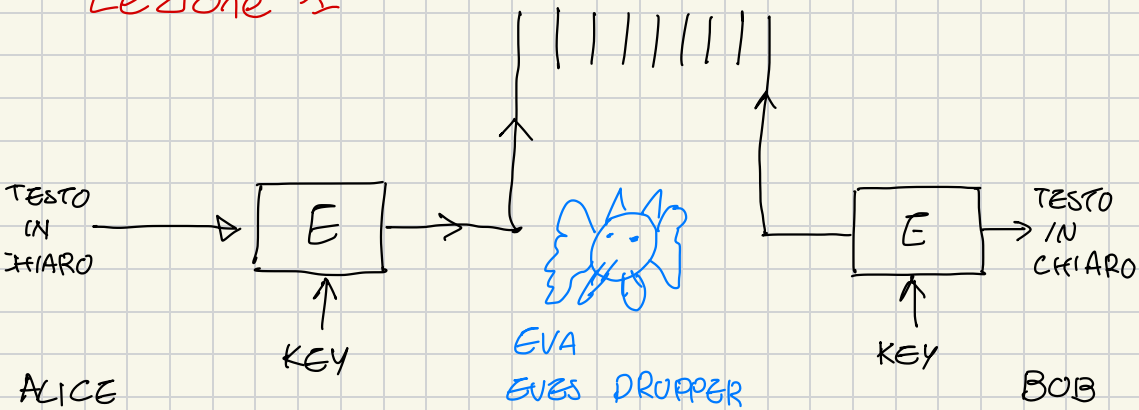
Lezione 1



NECESSARIO COSTRUIRE UN PROTOCOLLO INATTACCABILE
MA COSA VUOL DIRE 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} → numero

Ha bisogno di meno risorse

problemi → decisionali
problemi → operazionali
problemi → soddisfacibili → necessario provare i possibili valori

soddisfacibili → non si sa se esiste un algoritmo efficiente (polinomiale) in grado di risolverlo

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 x

→ similare all'idea di riduzione funzionale (FONDAMENTI)

GENERAZIONE NUMERI PSEUDOCASUALI

$$x_{i+1} = a x_i \bmod b$$

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

- OGGI NON SONO VENUTO A FABULARE PERCHÉ È MEGLIO DARE UNA BUONA LEZIONE DI CORRETTEZZA

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 infranto

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

Lezione 2

to hack → smanettare
↓
hacker → buoni, attaccano i sistemi e poi forniscono informazioni con cui sistemare

cracker → cattivi, vogliono attaccare i sistemi

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 riescono ad attaccarla

MAI INDURRE ALTRE PERSONE A VOLERCI ATTACCARE

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 avvolto 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 infrangere 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

CRITTOSISTEMA:

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. CIPHERTEXT: testo prodotto in output 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 trovare facilmente la chiave per poterli decifrare
2. Il mittente e il destinatario devono aver ricevuto copie delle chiavi segrete 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 $X = [X_1, X_2, \dots, X_M]$
- viene generata una chiave di cifratura $K = [K_1, K_2, \dots, K_J]$, 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 chiave 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 COPPIA 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

BU|ON|GI|OR|NO

CX NA IK NM AN

chiavi 25!

c'é 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 é possibile utilizzare diverse sostituzioni monoalfabetiche man mano che si procede attraverso il messaggio in chiaro. Il nome generico per questo approccio é cifrario di sostituzione polialfabetica. Tutte queste tecniche hanno le seguenti caratteristiche in comune:

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

c'é scorciatoia, possiamo nuovamente fare analisi delle frequenze

c'é sempre corrispondenza lettera uguale - lettera uguale

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

codifico 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 perche a distanze uguali, lettera uguale corrisponde a lettera uguale

PASWD PASWD PASWD PASWD

se nelle posizioni della stessa cobre 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**

↳ 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	K (key)	0	0	1	1
K=0	0	1	P (plaintext)	0	1	0	1
K=1	0	1	C (ciphertext)	0	1	1	0
	1	0					

(abbiamo applicato lo XOR)
↓
or esclusivo

L'algoritmo di decodifica è lo stesso di quello di codifica

Semplificazione del cifrario di Cesare su un alfabeto binario

$\bar{K}$ (vettore chiave)	0	1	1	0	1	0
$\bar{P}$ (vettore plaintext)	1	1	0	1	1	0
$\bar{C}$ (vettore ciphertext)	1	0	1	1	0	0

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

Stesso principio solo con due simboli

Ma siamo sicuri che guardando il ciphertext non so nulla?

$$C = X \oplus K$$

C → ciphertext, X → plaintext, $\oplus$ → XOR, K → chiave

K composta da bit scelti a caso, lunga quanto il messaggio. MONOUSO

X scelto a caso dall'utente

Sia P prob $[x=1]$

$$P_r[c=0] = P_r[x=0] \cdot P_r[k=0] + P_r[x=1] \cdot P_r[k=1]$$

probabilità che ciphertext sia 0

probabilità che plaintext sia 0

probabilità che Key sia 0

probabilità che plaintext sia 1

probabilità che Key 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 dal 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

Indovino la chiave con probabilità $\frac{1}{2^n}$

lunghezza della stringa

ENTROPIA

L'entropia misura il grado di 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

Shamon

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

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

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

valore negativo

valore positivo

Hartley è solo un caso particolare di Shamon

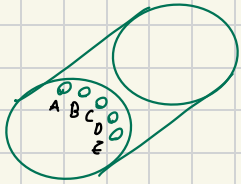
Lezione 3

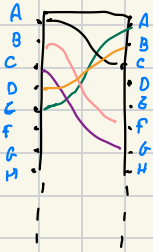
permutazione

$$\Pi : A \longrightarrow A$$

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

Applicazione composizionale di permutazioni corrisponde ad una sola permutazione

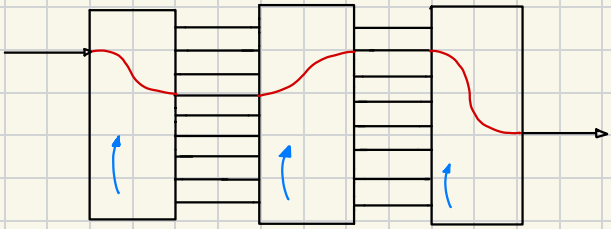




→ sono i contatti

MACCHINA ENIGMA (tentata di decodificare da Turing)

Enigma è una macchina utilizzata nella Seconda Guerra mondiale per cifrare messaggi. Decodifica molto complessa, vista la grande quantità di combinazioni possibili. Se ho una macchina che per essere decifrato ha bisogno di un tempo più alto del tempo di validità dei dati, allora posso dire che è sicur



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 di output

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

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

Known Ciphertext Attack (KCA)

Conoscendo il testo cifrato dobbiamo risalire al testo in chiaro

Known Plaintext Attack (KPA)

Conoscendo sia il testo in chiaro, sia il testo cifrato devo 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 ENCRIPTION STANDARD (DES)

Sistema con chiave 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 linearmente indipendenti sui bit della chiave

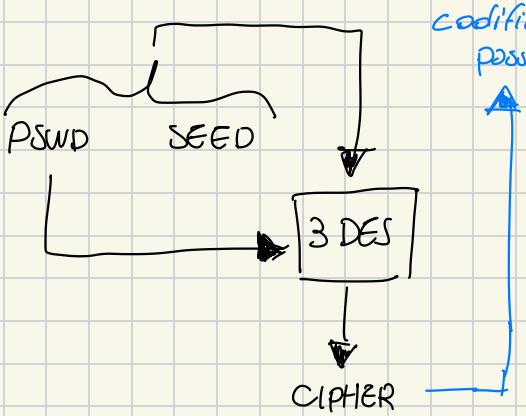
44 equazioni differenziali sui bit della chiave,

$(56-44) = 12$ bit scoperti, trovano la chiave

in 2^{12} tentativi
$\downarrow$
2048

Codifica password in Linux

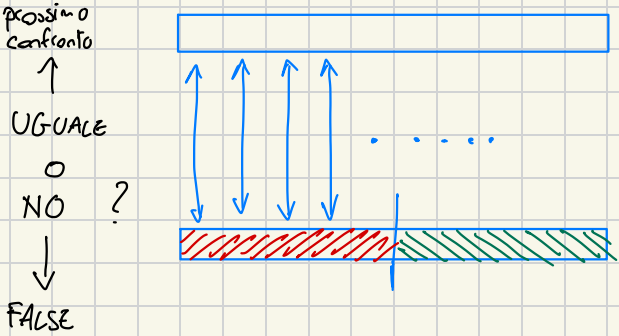
USERNAME : $\underbrace{\$x\,ab\,27\,c4}_{\text{codifica della password}}$ ..... : HOME : SEED :.....



$\hookrightarrow$ Ad ogni accesso codifica la password e la confronta con questa

IBM 360, mainframe super utilizzato in passato

Algoritmo di confronto fra stringhe



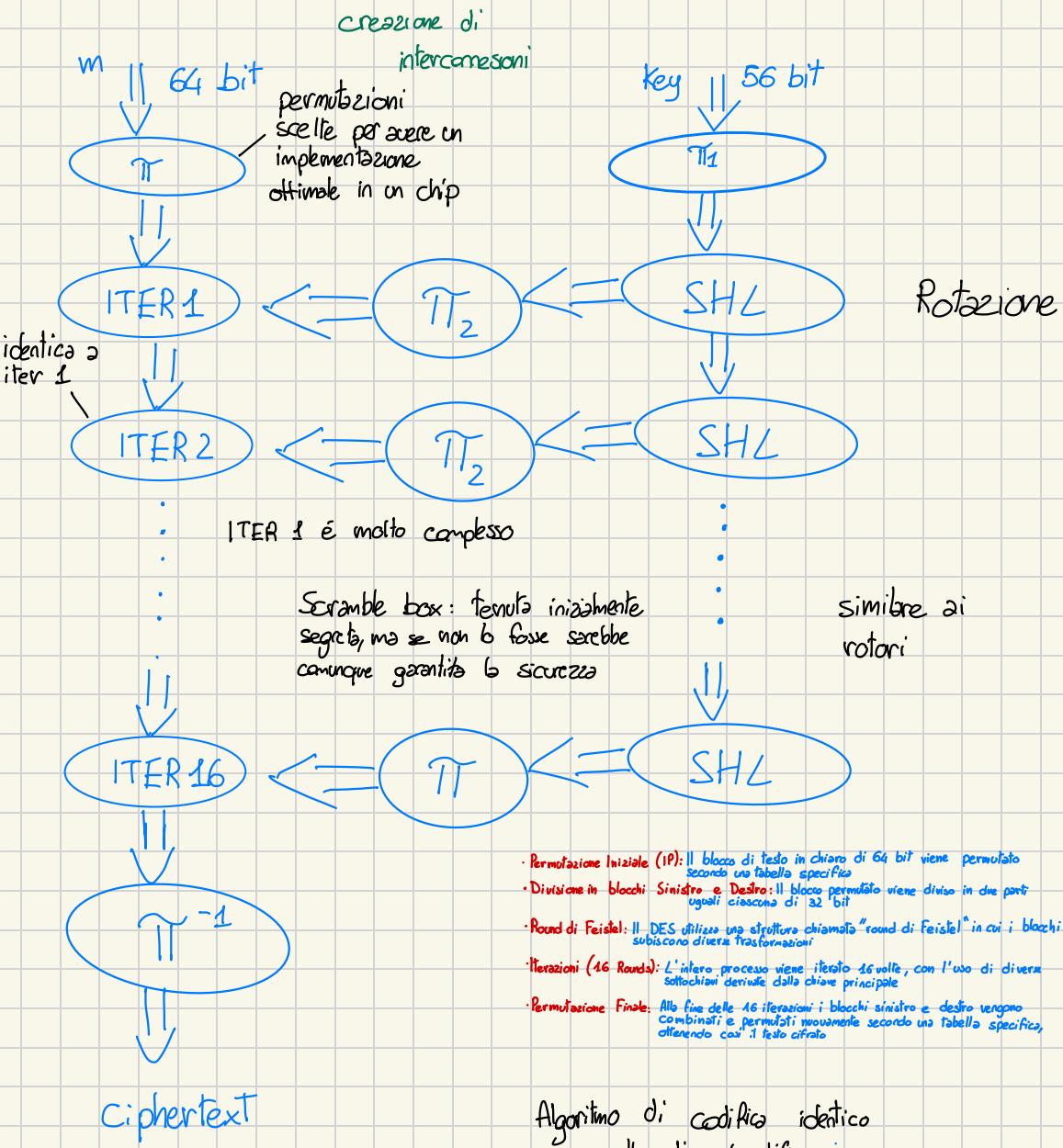
password inserite dall'utente vengono divise e scritte su due pagine diverse, di cui una fuori dalla cache

- se ho cache miss la parte di password nella pagina della cache è corretta
- 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 permutato secondo una tabella specifica
- **Divisione in blocchi Sinistro e Destro:** Il blocco permutato 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 sottochiavi derivate dalla chiave principale
- **Permutazione Finale:** Alla fine delle 16 iterazioni i blocchi sinistro e destro vengono combinati e permutati nuovamente secondo una tabella specifica, ottenendo così il testo cifrato

Algoritmo di codifica identico a quello di decodifica

Idea di Feistel

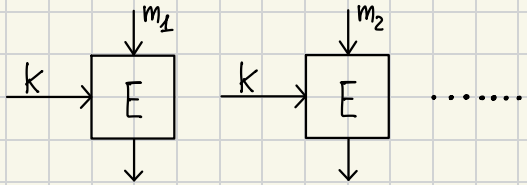
Come posso cifrare stream di blocchi?

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

Elettronic 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 |

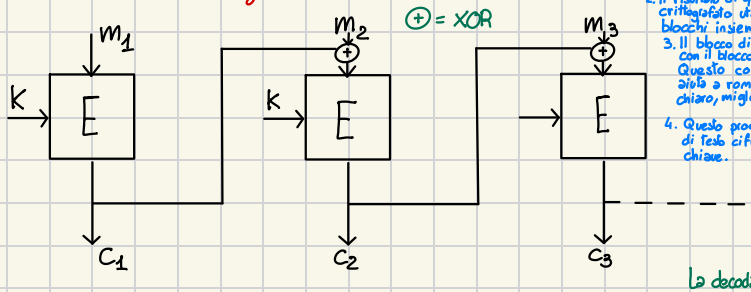
→ TRASF | 1 M E | A PIPPO |

- 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 come segue:

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 cifratura 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?

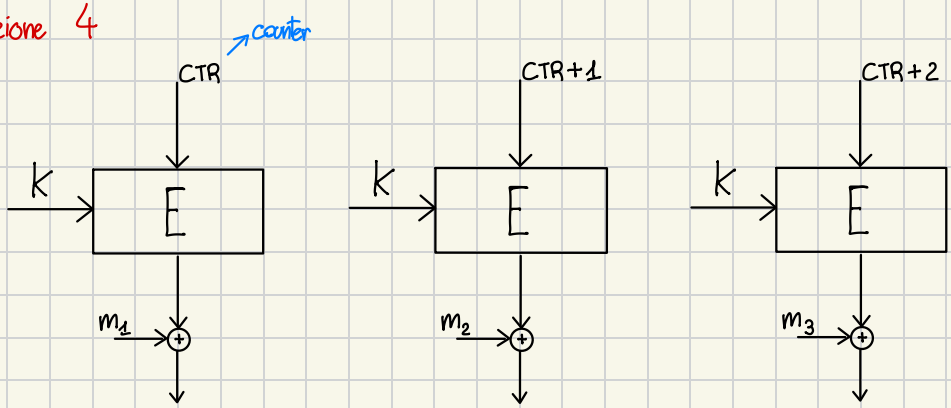
É un metodo simmetrico di cifratura quindi la procedura di decodifica é la medesima di quella di codifica

Come vengono gestiti i messaggi ripetuti?

Mi accorgo di messaggi ripetuti
Inserisco un blocco random a inizio messaggio → faccio XOR con valori casuali
per ogni messaggio ci sono $2^{\text{lungh blocco di num casuali}}$ possibili doppioni

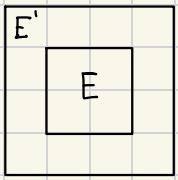
Tuttavia non riusciamo a spedire stream, vorrei trovare sistemi per spedire stream

Lezione 4



possiamo spedire i bit senza dover aspettare

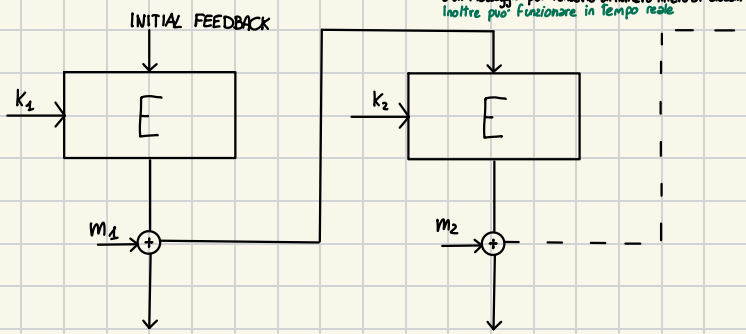
devo accordarmi sul valore di CTR



Se E non è attaccabile
Compongo E in E', se E' è attaccabile
allora lo era anche E

Ci riferiremo spesso a questo schema

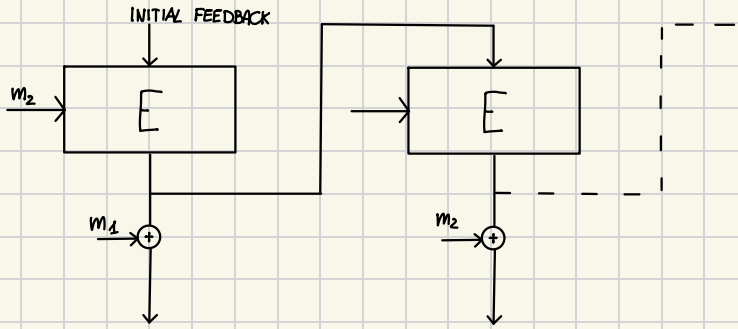
Cipher Feedback → cifrario a flusso: elimina la necessità di aggiungere padding a un messaggio per renderlo un numero intero di blocchi
Inoltre può funzionare in tempo reale



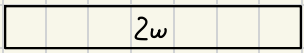
Assomiglia molto a cipherblock chaining tuttavia l'output del primo blocco viene messo in input al prossimo passaggio

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

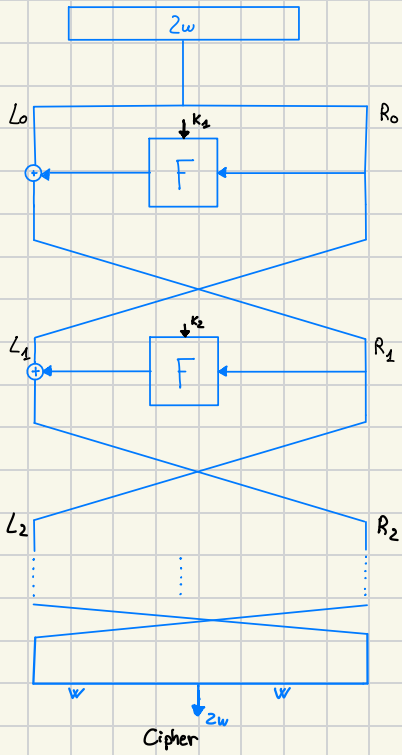
Output Feedback



FAISTEL → tecnica di cifratura a blocchi



L'input 2w é un numero pari di bit → viene diviso in due metá



A destra applico il blocco di Faistel → funzione di trasformazione che dipende da una sottochiave specifica del round

progettato per introdurre una complessità tale da rendere il processo di crittoanalisi complesso e costoso

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

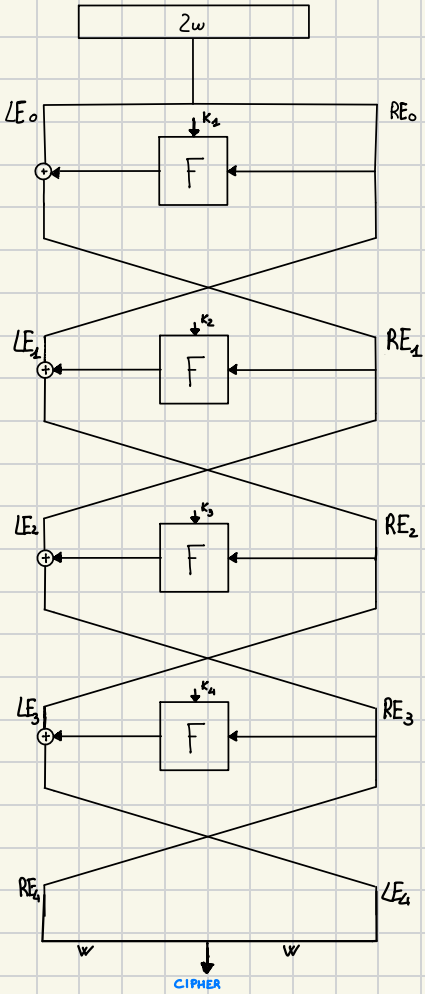
Le chiavi e le scatole di Feistel nel caso di DES sono i valori che vediamo, però in realtà lo schema di Feistel funziona qualsiasi sia la scatola F

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



Cosí facendo ottengo sempre i risultati dello schema precedente

$$LD_1 = RD_0 = LE_4 = RE_3$$

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

$$\overset{!}{=} RE_4 \oplus F(RE_3, K_4)$$

$$\overset{!}{=} LE_3 \oplus \boxed{F(RE_3, K_4)} \oplus \boxed{F(R_3, K_4)}$$

XOR tra cose uguali dà zero

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

Cosí facendo ottengo sempre i risultati dello schema precedente

Diffusione

Struttura statistica del plaintext viene dissipata

$m_1, m_2, m_3, \dots, m_\ell$

lettera

$$y_n \longleftarrow \sum_{i=1}^{n} m_i \quad \text{MOD } 26$$

considero le lettere come numeri

$y_1, y_2, \dots, y_\ell$ ottengo un testo distribuito in maniera uniforme

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

Necessario che F sia robusta e ben progettata

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

Confusione

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

Quando cambio un bit mi aspetto mediamente almeno la metà cambi

Il problema è scambiare la chiave

chiave pubblica - chiave privata

Crittografia moderna → si cerca di partire da problemi difficili su cui hanno lavorato in tanti
↓
si parte dalla teoria dei numeri

La funzione F introduce confusione nel testo in chiaro, in modo che la relazione tra 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 conosco la relazione tra 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

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

TEORIA DEI NUMERI → studio degli interi e delle loro relazioni

$(A, *)$

$* : A \times A \to A$

Associativa	$\forall a,b,c \in A$	$(a*b)*c = a*(b*c)$
Neutro	$\exists e \in A \ \forall a \in A$	$a*e = e*a = a$
Inverso	$\forall a \in A \ \exists a^{-1}$.	$a*a^{-1} = a^{-1}*a = e$

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

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

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

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

$\mathbb{N} \ op \ \mathbb{N} \to \mathbb{N}$

funzione one way → facile da calcolare
→ difficile da invertire

Classe di equivalenza

$\mathbb{Z}_n \equiv_n$

insieme dei numeri modulo n

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

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

insiemi chiusi

esempio

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

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

Quando abbiamo una relazione di equivalenza possiamo costruire 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 denotano mediante parentesi quadre di un elemento di tale classe. Ogni singola classe di equivalenza può essere rappresentata da qualsiasi elemento della classe stessa

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

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

$10.00 \pmod{12} \equiv 22.00$

Lezione 5

$a/n = b$ con resto c

$a \bmod n$ risulta c

$a_1 \equiv a_2 \bmod n$ se e solo se il resto

congruo

$a = bn + c$

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

$a \equiv c / \bmod n$

il resto della divisione n é uguale al

resto della divisione per n

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

$[a]_n$ una classe di equivalenza é 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 é uno di quei valori

Algebra modulare

$Q \rightarrow$ razionali

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

SSE

$ad = bc$

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

$Z_{\equiv}$

modulo

$[a] + [b] \triangleq [a+b]$

classe 5 classe 4

Z_7 $5+4 = 9 = 2$

somma é classe 9

avendo Z_7 $9 \bmod 7 = 2$

0 é l'elemento neutro

Proprietá 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

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

$(2n, +)$

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

$(Z_n, *)$

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

$1 \times$

$(Z_n^*, *)$

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

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

$xa + yb = MCD(a, b)$

a, n

$xa + yn = 1$

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

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

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

- Commutativa, Associativa, Elemento neutro

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

$b < a$		
MCD	a	b
MCD	$(a-b)$,	a
1b	$a-b$	a

$xa = 1 - yn$

Teorema di Eulero

Per ogni a, b $\exists x, y$ $ax + by = mcd(a, b)$

Se $a \in Z_n^*$ allora $mcd(a, n) = 1$ 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

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

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

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

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

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

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

$z = g \quad g^2 \quad g^3 \quad g^4 \quad g^5 \quad g^6 \quad g^7 \quad g^8 \quad g^9 \quad g^{10}$

g = 2 in questo esempio

logaritmo discreto è l'esponente da dare alla base per ottenere un numero

Continuando a moltiplicare g per se stesso, non arriverò ad un qualsiasi n generica, poiché il gruppo è finito.

Quindi supponiamo che arrivi a $g \otimes g \otimes g \otimes \ldots\ldots \otimes g = g^{|G|}$ e che l'esponente successivo sia $g^{|G|+1}$. Per il pumping lemma avrò sicuramente almeno un elemento ripetuto

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

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

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

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

TEOREMA

$$g^{\varphi(n)} = 1$$

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

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

$g^i \quad g^{i + \frac{p-1}{2}} \rightarrow -1$

$\Downarrow$

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

$g^{2i} \cdot g^{p-1} = g^{2i}$

cardinalità del gruppo

Generatore primi

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

In generale un numero primo non puó essere scomposto in fattori, quindi:

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

Esempio

- 1
- $2 \cdot 2 = 4 \cdot 2 = 8 \equiv 1 \pmod 7$
- $3 \cdot 3 = 9 \equiv 2, 2 \cdot 3 = 6, 6 \cdot 3 = 18 \equiv 4, 4 \cdot 3 = 12 \equiv 5, 5 \cdot 3 = 15 \equiv 1 \pmod 7$
- $4 \cdot 4 = 16 \equiv 2, 2 \cdot 4 = 8 \equiv 1$
- $5 \cdot 5 = 25 \equiv 4, 4 \cdot 5 = 20 \equiv 6, 6 \cdot 5 = 30 \equiv 2, 2 \cdot 5 = 10 \equiv 3, 3 \cdot 5 = 15 \equiv 1$
- $6 \cdot 6 = 36 \equiv 1 \pmod 7$

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

Densitá dei numeri primi

La densitá dei numeri primi é inversamente proporzionale al numero di bit che compongono il numero

Supponiamo di avere un numero casuale n di K bit, allora la probabilitá che n sia primo é $\frac{1}{K}$

Supponiamo di voler comporre un numero n di K bit, utilizzo un algoritmo che mi generi 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 verifico che il numero scelto sia primo. Statisticamente circa la metá dei numeri scelti sono testimoni del fatto che n non sia primo. Se il test fallisce e dice che il numero n non é primo, allora termino. Se il test ha esito positivo allora scelgo un altro numero a e ripeto il test di primarità. Se tutte le volte che scelgo un numero a il test ha esito positivo, allora la probabilitá di accettare la primarità di n é $\frac{1}{2^t}$, dove t é 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 enumerano l'intero gruppo, ci sará una potenza x che mi permette di ottenere a:

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

L'oggetto x é detto logaritmo discreto di a in base g. Trovare il logaritmo discreto di un numero <u>é un problema difficile</u>, 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. Difficoltá aumenta con dimensione del gruppo

Numero Quadrato

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

$x \otimes x = a$

Quindi esiste una radice quadrato di a in G. Gli elementi in $\mathbb{Z}_p^*$ sono $p-1$, ma solo la metá 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 dia il numero dell'insieme, il numero risultante sará un quadrato.

Esempio

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

- $1 \cdot 1 = 1$ quindi 1 é un quadrato
- $2 \cdot 2 = 4$ quindi 4 é un quadrato
- $3 \cdot 3 = 9 \equiv 2$ quindi 2 é un quadrato
- $4 \cdot 4 = 16 \equiv 2$ quindi 2 é un quadrato
- $5 \cdot 5 = 25 \equiv 4$ quindi 4 é un quadrato
- $6 \cdot 6 = 36 \equiv 1$ quindi 1 é un quadrato

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

Teorema

Tutti gli elementi g^{2i} sono quadrati in $\mathbb{Z}_p^*$

Se consideriamo un generatore g di $\mathbb{Z}_p^*$ e generiamo l'intero gruppo, é 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 $g^{2i+\frac{p-1}{2}}$

Infatti
$$g^{2i} = \begin{cases} g^{2i} \\ g^{2i+\frac{p-1}{2}} \end{cases}$$

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

So che le radici quadrate sono anche

$$g = \begin{cases} g^{i} \\ g^{-i} \end{cases}$$

Quindi elevare un generatore alla cardinalitá del gruppo mi dá 1, ma elevarlo alla metá della cardinalitá del gruppo mi dá -1

Con questa osservazione possiamo costruire un algoritmo che mi permette 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 efficente non preclude la possibilitá 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

cardinalità del gruppo

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

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

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

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 efficenti per calcolare il logaritmo discreto, so 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 significato di un numero in maniera efficente, 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

$\mathbb{Z}_n^*$ $\qquad n = p \cdot q$

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

x^b

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

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

Iterative Squaring

Con il simbolo di Legendre possiamo calcolare in maniera polinomiale, ma non in maniera efficente
Se eleviamo un numero a per b non operiamo con un algoritmo efficente.
In $\mathbb{Z}_p^*$ è possibile calcolare in tempo polinomiale un'esponenziale 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:

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

Calcoliamo iterativamente i quadrati ottenuti e moltiplichiamo quelli che ci servono

$$x^{2^5} \quad x^{2^4} \quad x^{2^3} \quad x^{2^2} \quad x^{2} \quad x$$

Facendo 5 moltiplicazioni ho trovato x elevato al peso dei vari bit

$$x^{2^5} \quad x^{2^4} \quad x^{2^3} \quad x^{2^2} \quad x^{2} \quad x$$

Faccio 2K operazioni al posto di 2^K

L'algoritmo per moltiplicare due numeri a K bit la complessità è K^2

numero grande

SHIFT A DESTRA IN BINARIO → DIVIDO per 2

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

Procedure Iterative Squaring(a,b)

tiene i quadrati iterati

```
Exp (x, b)
    y ← 1
    t ← x
    while b ≠ 0
        if (b % 2 != 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 indesiderata
per far diventare polinomiale
inseriamo n

```
Exp (x, b, n)
    y ← 1
    t ← x
    while b ≠ 0
        if (b % 2 != 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 $n = pq$

TEOREMA

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

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

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 $\mathbb{Z}_n^*$, chiama funzione di Eulero $\varphi(n) \triangleq |\mathbb{Z}_n^*|$

Ma se ho n e non p e q?

Nessuno conosce 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

cardinalità

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

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

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

Teorema

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

Posso 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. Basta calcolare il simbolo di Legendre rispetto a p e q e so il risultato.

Se non conosco la fattorizzazione di n non conosciamo 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

Adesso dimostriamo (probabile security) che se riuscissimo a calcolare le radici quadrate allora riusciremo a fattorizzare n

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

ALLORA $MCD(x+y, n) = p$ o q

Dimostrazione

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

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

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

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

sia $p \mid (x+y)$

questo divide

è possibile che $q \mid (x+y)$?

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

ALGORITMO FATTORIZZA (n)

$a \in_R Z_n^*$

prendi uniformemente un elemento di Z_n^*

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

Sia x la radice quadrata di a^2

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

if $x \neq \pm a$ ret 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 stima sia similare a $\log n = K$

numero di bit

→ Conseguentemente $\frac{1}{K}$ volte trovo il numero primo

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

Come capiamo se un numero è quadrato o meno in $\mathbb{Z}_n^*$

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

è una generalizzazione del simbolo di Legendre

o vale 1 o vale -1

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p}\right)\left(\frac{a}{q}\right)$$

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

$$\left(\frac{a_1 a_2}{n}\right) = \left(\frac{a_1}{n_1}\right)\cdot\left(\frac{a_2}{n_2}\right)$$

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 dice 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

Trucco → fai le cose a caso → se ok va bene
↓
se no ok rifai

La quadraticità di un numero con simbolo di JACOBI 1 è un altro dei problemi difficili

$x \cdot y$

sia x quadrato e sia y quadrato → xy quadrato

$\left(\frac{xy}{p}\right)$ un numero per essere quadrato deve essere in Z_p^* e Z_q^*

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

Identico con Z_q^*

Un numero è quadrato in Z_n^* se e solo se è un quadrato in Z_q^* e Z_p^*

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

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

Se x è non quadrato y non quadrato con entrambi simbolo 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 Z_p^*

Idem su Z_q^*

Quindi su 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 al quadrato in Z_n^* è ONE WAY TRAPDOOR

↓ facile da calcolare difficile da invertire

se sappiamo fattorizzare però 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 $n = p \cdot q$, visto che x e y sono radici quadrate di uno stesso numero sappiamo che $x^2 \equiv y^2 \mod n$. Quindi:

$$x^2 \equiv y^2 \mod n \Rightarrow x^2 - y^2 \equiv 0 \mod n$$

$$\Rightarrow (x-y)(x+y) = K \cdot n \quad \text{per qualche } K$$

Supponiamo che p divida $x+y$ ma è possibile che q divida $x-y$? Se fosse possibile allora n divide $x+y$, ma ciò vorrebbe dire che $x+y \equiv 0 \mod n$, ovvero $x \equiv -y \mod n$ e ciò è assurdo perché $x \not\equiv \pm y$. Quindi $\mathrm{mcd}(x+y, n) = p$.

Supponiamo che q divida $x+y$ ma è possibile che p divida $x-y$? Se fosse possibile allora n divide $x+y$, ma ciò vorrebbe dire che $x+y \equiv 0 \mod n$, ovvero $x \equiv -y \mod n$ e ciò è assurdo perché $x \not\equiv \pm y$. Quindi $\mathrm{mcd}(x+y, n) = q$.

Supponiamo che p non divida $x+y$ e che q non divida $x+y$, 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 $x \equiv y \mod n$ e ciò è assurdo.

Capitolo 2. Teoria dei numeri 40

Supponiamo che p divida $x+y$, ma è possibile che q divida $x-y$? Se fosse possibile allora n divide $x+y$, ma ciò vorrebbe dire che $x+y \equiv 0 \mod n$, ovvero $x \equiv -y \mod n$ e ciò è assurdo perché $x \not\equiv \pm y$. Quindi $\mathrm{mcd}(x+y, n) = p$.

Supponiamo che q divida $x+y$, ma è possibile che p divida $x-y$? Se fosse possibile allora n divide $x+y$, ma ciò vorrebbe dire che $x+y \equiv 0 \mod n$, ovvero $x \equiv -y \mod n$ e ciò è assurdo perché $x \not\equiv \pm y$. Quindi $\mathrm{mcd}(x+y, n) = q$.

Supponiamo che p non divida $x+y$ e che q non divida $x+y$, 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 $x \equiv y \mod n$ 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 $A \in \mathtt{PPT}$ (*probabilistic polynomial time*) che calcoli la radice quadrata di un numero in $\mathbb{Z}_n^*$. Allora:

```
procedure FACTORIAL(n)
    x ∈_R Z_n^*
    y ← A(x^2)
    z ← mcd(x − y, n)
    if z ≠ n then return z
    else
        return Factorial(n)
    end if
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 di x è $\frac{1}{2}$, 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 Diffe-Helman (crittografia a chiave pubblica-chiave privata)

Z_p^*, g → decisi dal National Institute for Security and Tecnology

p: numero primo

Z_p^*: gruppo zeta p star

g: generatore del gruppo

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

A (authority) sceglie $x \in_R \{1, \dots\dots, p-1\}$ → chiave privata

elementi di Z_p^*

calcola $g^x \bmod p$ → chiave pubblica

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

B sceglie $y \in_R \{1, \dots\dots, p-1\}$

↳ numero

cardinalità del numero $= -1$

$p-1$

calcola $g^y \bmod p$

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

In un sistema a chiave pubblica abbiamo i seguenti algoritmi

- Algoritmo di generazione delle chiavi $G: 1^k \to (P_k, S_k)$ — Dove 1 è il security parameter che indica la lunghezza della chiave
- Un algoritmo di encription: $\mathcal{E}: m, P_k \to \mathcal{E}(m, P_k)$
- Un algoritmo di decription: $D: C, S_k \to D(C, P_k)$

Ovviamente vale la seguente relazione

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

Il sistema è sicuro perché calcolare $g^{xy} \bmod p$ è computazionalmente intrattabile. 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

$(g^y)^x$, $(g^x)^y$ → g^{xy} → Il numero risultante è il medesimo

↳ questa è la chiave utilizzata da A e B per comunicare

Decrypbre un messaggio a partire dal testo cifrato senza chiave privata è computazionalmente intrattabile

→ Hanno chiave comune ma 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-riduzione e dimostrare che il problema di riferimento è legato al calcolo del logaritmo discreto

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

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

Ipotesi di Diffe-Helman, 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

$\mathcal{E}$ $\quad b = \{0, 1\}$

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

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

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

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

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

$$P_r\left[B(m, n, \ell) = b\right] = \frac{1}{2}$$

Ho un bit b sparato a caso

probabilità di indovinare il bit b

chiunque abbia una potenza di calcolo polinomiale non riesce a notare b 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 $\frac{1}{2}$

↳ nuovo

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

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

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

↳ sparando x, y e z a caso sto sparando k bit a caso per ciascuno dei tre se indovino m, n, l calcolo le due triple di b e vedo quali delle due ho indovinato altrimenti se x, y e $z \neq m, n, l$ 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 $\quad Pr[B(m,n,l)=b]$

Definitivamente non finisce → ovvero non importa all'inizio ma da un certo punto in poi

$$\forall_B \forall_C \exists \bar{k} \forall_{k>\bar{k}} \quad \left|P - \frac{1}{2}\right| < \frac{k}{-c}$$

costante

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

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

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

↳ voglio che sia piccolo quindi devo avere poco vantaggio

numero di bit

Se ho una chiave di k bit piccola riesco a indovinare la soluzione, perciò considero le chiavi più lunghe

Rivest Shamir Adleman - RSA

g^x è una funzione one-way. Il protocollo di Diffie-Helman è sicuro se e solo se esiste una funzione one-way.
↳ meglio se trapdoor

PUB KEY	n, e
PRIV KEY	n, d

p, q primi a caso con $k/2$ bit

$n = pq$

$(p-1)(q-1)$ numero di Eulero

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

↳ numero coprimo

$e \in_R \mathbb{Z}^*_{\varphi(n)}$

$d = e^{-1} \bmod \varphi(n)$ tale che $ed \equiv 1 \bmod \varphi(n)$

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

Encription algorithm

$E : m, (n, e) \longmapsto m^e \bmod n$

Decription Algorithm

$D : c, (n, d) \longmapsto c^d \bmod n$

Funzionamento

$k\varphi(n) + 1$

$$m^{ed} = m$$

↳ ciphertext (CHIAVE PUB (n,e))

↳ decodifica (CHIAVE PVT) (n,d)

$$= (m^k)^{\varphi(n)} \cdot m$$

$$= m \quad 1$$

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

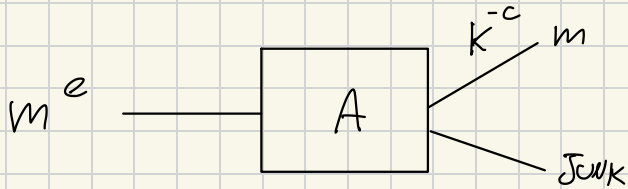
L'unico modo per calcolare la radice e-esima è calcolare d e decifrare il messaggio, quindi d è l'informazione trapdoor

Infatti $d \cdot e$ è congruo a 1 modulo $\varphi(n)$, quindi $d \cdot e = k\varphi(n) + 1$

Per il teorema del resto cinese, $m^{k\varphi(n)+1} \equiv m$, e non solo per gli elementi di $\mathbb{Z}^*_n$

La funzione one-way è la funzione di codifica, quindi m^e, l'inversa di m^e è la radice e-esima ovvero $m = c^{1/e}$ ma per calcolare la radice e-esima ad oggi non esiste un'algoritmo efficiente

Riesco a calcolare d? no perché non posso calcolare $\varphi(n)$ da n poiché se n è primo so che è $n-1$ ma altrimenti è $(p-1)\cdot(q-1)$ dove $n = p \cdot q$ e quindi dovrei essere in grado di fattorizzare 1 (primo, primo)



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

Sarebbe estremamente pericoloso perché in K^{-c} volte posso arrivare alla soluzione

funziona biettiva, ho un elemento a caso di Z_n^*

$c \cdot x^e$ → elemento a caso di Z_n^*

funziona biettiva, ho un elemento a caso di Z_n^*

$x \in_R Z_n^*$

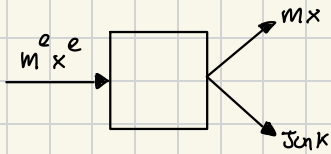
$$\sqrt{c \cdot x^e} = \sqrt{m^e x^e} = \frac{m\,x}{x} = m$$

c: ciphertext ovvero m^e

x: scelto da noi

Dalla soluzione al problema trasformato trovo la soluzione

Così facendo ho



Lezione 8

Non riuscire a decodificare

Se due agenti inviano allora i mex sono uguali?

MALLEABILE: con due mex in chiaro, codifica legata

radice n-esima

non si usa il modulo, decriptiamo facilmente

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

Protocolli dimostrabilmente sicuri?

Ideali, codifica di un singolo bit

Dispendioso

$G: 1^K \mapsto (S,P)$

$E: P, m \mapsto E(m,P)$

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

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

Algoritmo di Generazione

Scegliamo $n = p_1 p_2$ (primi) rappresentabili con $K/2$ bit $p_1\ p_2 \in_R$ PRIMI (scelti a random)

$y \in_R$ non quadrato con $J=1$, sparo bit a caso

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

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

$P_K = (m, y)$ — non quadrato mi permette di costruire

$S_K = (p_1, p_2)$

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

Encription

$$b \longmapsto \begin{cases} x^2 & b=0 \quad x \in_R \text{QUADRATI} \\ x^2 y & b=1 \quad x \in_R \text{QUADRATI} \end{cases}$$

L'algoritmo di codifica prende un bit b, sia $x \in_R \mathbb{Z}_n^*$, b è 0 allora $c = x^2 \bmod n$, altrimenti $c = xy \bmod n$. x^2 è un quadrato casuale di $\mathbb{Z}_n^*$, mentre xy^2 è un non quadrato con simbolo di Jacobi. Se prendo un quadrato con simbolo di Jacobi 1 e 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 ottengo un non quadrato casuale con simbolo di Jacobi 1 distribuito uniformemente tra tutti i non quadrati con simbolo di Jacobi 1.

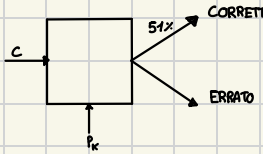
COSA VUOL DIRE ATTACCARE IL PROTOCOLLO?

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

Decriptare è $1/2^2$, la probabilità di indovinare

se inferiore inverto il risultato

Se riuscissi ad indovinare con probabilità leggermente superiore



Supponiamo per assurdo che esista

aurei rotto il protocollo

il divario è maggiore con tanti esperimenti

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 cyperteXt valido

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

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

Sicure222

$$\forall c\ \exists \bar{k}\ \forall k > \bar{k} \quad \left| P[\text{successo}] - \frac{1}{2} \right| > k^{-c}$$

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

$p \mapsto$ prob. di successo

$P[\text{MAGGIORANZA DI EXP ABBIA SUCCESSO}] = \sum_{i=\frac{n}{2}+1}^{n} P_K[\text{i esp di successo}]$

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

esponenzialmente piccolo in n

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

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

con un numero polinomiale di esperimenti mi avvicino a 1.

security param.

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

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

$$2^{-2cn\left(p-\frac{1}{2}\right)^2} \leq 2^{-k}$$

$$-2cn\left(p-\frac{1}{2}\right)^2 \leq -k$$

$$n \geq \frac{k}{2c\left(k^{-c}-\frac{1}{2}\right)^2}$$ Quindi polinomiale

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

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

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 vicino a 1, dove l'esponenzialmente è in funzione di n.

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

Dove ε è il vantaggio

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

$$e^{-2c\varepsilon^2 n} < 2^{-k}$$

$$-2\varepsilon^2 n < c' - k$$

$$n > \frac{c'-k}{2\varepsilon^2}$$

Se ε 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 piccolo, scegliendo una quantità di esperimenti polinomiale in esso

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

Nel momento in cui ε è $\bar{k}^{-c}$, allora n (dove n è il numero di esperimenti) è polinomiale in $\bar{k}$.

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

Attaccante con vantaggio polinomiale

Costruiamo esperimenti indipendenti sull'analisi del vantaggio polinomiale

c, costruisco L esperimenti indipendenti

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

costruisco $cR_1^2, cR_2^2, \dots, cR_L^2 \longleftarrow$ Input casuale, ho reso casuale il problema

Avranno stessa 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%

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

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

Allora la trasformazione non va bene, serve distribuzione uniforme su $J=1$

Scegliamo anche bit a caso.

$b_1, b_2, \dots, b_L \longleftarrow$ solo per aver distribuzione

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

y non quadrato con J = 1

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

y_i é distribuito uniforme

Risponde al problema opposto, quindi la complemento se b_i é 1 → perché ho invertito quadraticità

Prendo la maggior parte delle risposte

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

$b_1 \ldots\ldots b_\ell$

$E(b_1)E(b_2)\ldots\ldots E(b_\ell)$

é un electronic code book in cui si compongono i singoli bit

MALLEABILE posso scambiare i bit → proviamo a ignorare questo problema, vorremmo che dal cypertext non si possa risalire ad alcuna informazione binaria sul plaintext

Distinguisher $D \in PPT$

io distingo 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 m_1 dato il cipherfext con il security parameter che vale k

Un sistema di cifrica nasconde m_1 e m_2 a D quando

$$\forall c \ \exists \bar{k} \ \forall k > \bar{k} \ \left| P_k^{D,m_1} - P_k^{D,m_2} \right| < k^{-c}$$

probabilità che il distinguisher riconosca m_2 dato il cipherfext

Sto dicendo che non potrò mai usare il distinguisher per capire se il cipherfext restituito é quello di m_1 o quello di m_2

Ovvero:

$$\left| P_k^{D,m_1} - P_k^{D,m_2} \right| < k^{-\omega(1)}$$

qualsiasi numero naturale

$\phi \quad \{\phi\} \quad \{\phi, \{\phi\}\} \quad \ldots\ldots \quad \omega$ — primo numerale infinito

$\{\omega, \{\omega\}\} \ldots\ldots$ avanti all'infinito

"La nostra vita é polinomialmente piccola"

E nasconde a D informazioni se per ogni m_1 e m_2 E nasconde m_1 e m_2 a D

E nasconde se per ogni $D \in PPT$ E nasconde m_1 e m_2 a D

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

Dimostrazione

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

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

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

$m_1 = \alpha_1 \alpha_2 \ldots\ldots \alpha_\ell$

$m_2 = \beta_1 \beta_2 \ldots \beta_\ell$

m_2 è m_1 cambiando un bit alla volta

Definisco $\forall i \in \{0, \ldots, \ell\} \ \ m(i) = \beta_1 \ldots\ldots \beta_i \ \alpha_{i+1} \ldots\ldots \alpha_\ell$

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

$m(\ell) = m_2 \qquad P(\ell) = P_k^{D,m_2}$

$P(i) = P_k^{D,m(i)}$ ovvero la probabilità che il distinguisher restituisca 1 se viene dato in input una codifica del messaggio m_i

capacità del distinguisher di sapere qualcosa

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

disuguaglianza di Wallstrass

↳ Sappiamo che $\cancel{P(0) - P(1)}$, $\cancel{P(1) - P(2)}$, $\cancel{P(\ell-1) - P(\ell)}$ resta $P(0) - P(\ell)$

Quindi:

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

Avendo la somma di numeri che eccede un determinato valore allora so che esiste almeno un elemento che è maggiore o uguale della media

$$\exists i \in \{0, \ldots, \ell-1\} \text{ t.c } |P(i) - P(i+1)| \geq \frac{k^{-c}}{\ell} \geq k^{-c'}$$

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

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

$m(i+1) = \beta_1 \ldots \beta_i \ \boxed{\beta_{i+1}} \ \alpha_{i+2} \ldots\ldots \alpha_\ell$

Supponiamo senza perdita di generalità che $\alpha^{i+1} = 0$ e $\beta^{i+1} = 1$. Sia z un elemento di $\mathbb{Z}_n^*$ con $\left(\frac{z}{n}\right) = 1$, quindi z potrebbe essere la codifica di uno 0 o di un 1.

Dato z viene costruito $E(\beta_1)E(\beta_2)\ldots\ldots E(\beta_i)\, z\, E(\alpha_{i+2})\ldots\ldots E(\alpha_\ell)$ e viene lanciato l'algoritmo D sul risultato. La probabilità con cui D restituisce 1 è $P(i)$ 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_i, ovvero applicando E a tutti i bit del messaggio. Avendo però aggiunto z in mezzo al messaggio non ho codificato secondo l'algoritmo che il distinguisher si aspetta, quindi bisogna codificare il messaggio $m(i)$ in maniera corretta:

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

non è un quadrato casuale

$$E(\beta_1)E(\beta_2)\ldots E(\beta_i)\,(x^2 z)\,E(\alpha_{i+2})\ldots\ldots E(\alpha_\ell) \quad \text{con } x \in \mathbb{Z}_n^*$$

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

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

Se si riceve la codifica di un bit scelto a caso, con quale probabilità si riesce a distinguere se è 0 o 1? Si utilizza il risultato del distinguisher D come tentativo

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

Se sottraiamo $\frac{1}{2}$ otteniamo:

$$\begin{aligned} P[\text{non indovinare}] &= \frac{1}{2} + \frac{1}{2}\cdot(P_1 - P_0) - \frac{1}{2} \\ &= \frac{1}{2}(P_1 - P_0) \\ &= \frac{1}{2}\cdot|P(i) - P(i+1)| \\ &\geq \frac{1}{2}\cdot k^{-c'} \end{aligned}$$

→ L'algoritmo ha quindi un vantaggio di almeno $\frac{1}{2}\cdot k^{-c'}$, e quindi tale algoritmo è un attaccante.

ho costruito un distinguisher tra quadrati e non quadrati

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

Distinguisher

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

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

Lezione 10

Abbiamo il nostro algoritmo A che indovina

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

$$\parallel$$

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

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

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

$$P_r[x] = P_r[b=0]\,P_r[x \mid b=0] + P_r[b=1]\,P_r[x \mid b=1]$$

$$= \frac{1}{2}\left(1 - \overset{A}{\overset{\parallel}{P_k^{D,0}}}\right) + \frac{1}{2}\overset{A}{\overset{\parallel}{P_k^{D,1}}}$$

Supponiamo di indovinare

without loss of generality

$$P_r[x] > \frac{1}{2} + k^{-c} \quad \text{w.l.o.g}$$

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

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

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

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

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

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

Costruire un distinguisher è equivalente a costruire qualcosa che indovina

Abbiamo costruito il primo crittosistema dimostrabilmente sicuro

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

→ è un problema, spreco di banda e risorse → vorremmo codificare un bit con un solo bit → impareremo a costruire numeri pseudo-casuali crittograficamente sicuri da utilizzare come base per simulare one time pad (fora' da chiave)

→ nessuno con potenza di calcolo polinomiale può notare la differenza

Lancio della moneta in rete → problema difficile da simulare

Abbiamo due agenti A, B — $COIN_A, COIN_B$ (variabili dove Alice e Bob salvano il valore)

A, B ONESTI — $COIN_A = COIN_B$ — $P_r[COIN_A = 0] = \frac{1}{2}$

A ONESTA, B DISONESTO — $\left|P_r[COIN_A = 0] - \frac{1}{2}\right| < k^{-\omega(1)}$

A DISONESTA, B ONESTO — $\left|P_r[COIN_B = 0] - \frac{1}{2}\right| < k^{-\omega(1)}$ → discostamento 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 $k^{-\omega(1)}$

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

Lancio della moneta con residuo quadratico

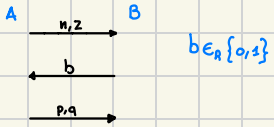
Ci basiamo sulla difficoltà di stabilire se un numero è quadrato o meno in $\mathbb{Z}_p^*$ #

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

A p,q primi a caso

$n = p \cdot q$

$z \in_R$ con $\left(\frac{z}{n}\right) = 1$



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

$COIN_A = COIN_B = b \oplus \underbrace{IS_SQUARE(z)}_{\frac{1}{2}}$

Sostanzialmente A lancia una moneta (sceglie a caso fra un quadrato e un non quadrato)# e B sceglie un bit B a caso e lo invia ad A. Quando B riceve z non sa qual è il risultato del lancio della moneta, perché non sa 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 z è stato campionato secondo una misura casuale, quindi z è un quadrato con probabilità $\frac{1}{2}$ e un non quadrato con probabilità $\frac{1}{2}$. Anche b è stata 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 interi parametri dell'algoritmo, che sarà uguale per entrambi seguendo la formula: $b \oplus isSquare(z)$

Se B è disonesto, allora può calcolare il 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 alla quadraticità 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

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

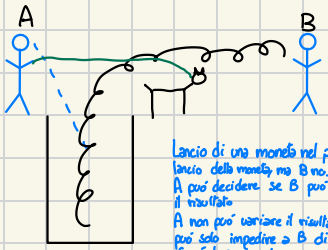
Alice ha quindi la capacità di alterare la probabilità

↳ PROB successo va a $\frac{3}{4}$

Il protocollo non è sicuro

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

Lancio di moneta 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 partecipare o meno al processo, non gli interessa il risultato del lancio della moneta, ma interessa solamente che la probabilità del lancio sia almeno $\frac{1}{2}$

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 occorrenza del risultato che gli è favorevole, può decidere come risultato un valore che gli fornirebbe tale vantaggio. A non potrebbe non inviare p e q, in questo caso B ha un enorme svantaggio, perché dopo il secondo messaggio A conosce $coin_A$, ma B non conosce $coin_B$, se A invia il proprio messaggio a B, B è in grado di calcolare $Coin_B$ e quindi il risultato finale, ma se A sa il risultato vincente e il risultato atteso è sfavorevole, A non spedisce il terzo messaggio e B lancia la propria moneta che con probabilità $\frac{1}{2}$ è favorevole, ma con probabilità $\frac{1}{2}$ è sfavorevole, si crea una situazione sfavorevole a B

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

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 possano 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'invio dell'ultimo messaggio. Ciò 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 innesca una reazione a catena che porta A e B a non inviare nulla

Non esiste il protocollo per il lancio della moneta che soddisfi le condizioni dettate 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. 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 $\frac{1}{2}$

A		B
		Sceglie $p, q \in_R$ primi Calcola $n = p \cdot q$ Calcola e, d chiave p e p per RSA Codifica il segreto con RSA ottenendo un messaggio c
	$\xleftarrow{n, e, c}$	
Sceglie $x \in_R \mathbb{Z}_n^*$		
	$\xrightarrow{x^2}$	
		Calcola una radice quadrata di x, che sarà y
	$\xleftarrow{y}$	

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'opposta dell'altro, allora è in grado di fattorizzare 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 $\{g^0, g^1, \dots, g^{p-2}\}$. I due valori p e q sono condivisi tra A e B, prima dell'inizio del protocollo.

A		B
Sceglie $x \in_R \{0, \dots, p-2\}$ Calcola $z = g^x$		
	$\xrightarrow{z}$	
		Sceglie $b \in_R \{0, 1\}$
	$\xleftarrow{b}$	
	$\xrightarrow{x}$	

Il risultato è quindi $b \oplus \left(x < \frac{p-1}{2}\right)$. L'idea di fondo è che B quando sceglie b non è in grado di dedurre 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

Data 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 discreto, 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 sta 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 $x \in_R \{0, \dots, p-2\}$

$z \leftarrow g^x$

A		B
	$\xrightarrow{z}$	$b \in_R \{0, 1\}$
	$\xleftarrow{b}$	
	$\xrightarrow{x}$	

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

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. Se scegliessimo a caso il valore di tale predicato binario, la probabilità con cui un algoritmo riesce a calcolare il predicato conoscendo z è $\frac{1}{2} + \varepsilon$, con ε 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 è un quadrato, 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 principale di un quadrato in $\mathbb{Z}_p^*$, allora esiste un algoritmo efficente per il calcolo del logaritmo discreto in $\mathbb{Z}_p^*$

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 dovremmo seguire tre passaggi:

1. Mostrare 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. Mostrare che tale algoritmo funziona anche se la radice quadrata principale è calcolata con probabilità esponenzialmente vicina a 1
3. Mostrare 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)

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

```
procedure LSB(y)
    if y/p = 0 then
        return 0
    else
        return 1
    end if
end procedure
```

Idea: abbiamo macchina B

$$y = g^x \longrightarrow \boxed{B} \longrightarrow \begin{cases} 0 & \text{se } x < \frac{p-1}{2} \\ 1 & \text{altrimenti} \end{cases} \geq \frac{1}{2} + k^{-c}$$

$$\sqrt{y} \qquad g^{\frac{x}{2}} \qquad g^{\frac{x}{2} + \frac{p-1}{2}}$$

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(DiscreteLogarithm(y))
    if b = 1 then
        return y ← y·g^-1          ▷ imposto a zero il bit meno significativo
    end if
    y ← PSQR(y)                    ▷ scorri a destra di un bit
    return 2·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 $1-\varepsilon$. La probabilità che l'algoritmo DiscreteLogarithm invocato k volte fornisca sempre la risposta corretta è $(1-\varepsilon)^k$. L'algoritmo Discrete Logarithm può non funzionare, ma abbiamo 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 $(1-\varepsilon)^k$, allora il numero di volte che funziona in cui si dovrà lanciare l'algoritmo sarà $\frac{1}{(1-\varepsilon)^k}$. Se $\varepsilon \leq \frac{1}{2}$, allora $(1-\varepsilon)^k \geq \frac{1}{2^k}$

Se abbiamo un algoritmo che funziona con probabilità esponenzialmente vicina a 1, allora possiamo costruire un algoritmo che calcola il logaritmo discreto con probabilità almeno $\frac{1}{2}$, quindi se ripetiamo in media due volte l'algoritmo otteniamo 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 1?

Assolutamente sì, per farlo dobbiamo costruire tante istanze indipendenti dello stesso problema e prendendo il risultato osservato 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 y' tale per cui dalla risposta y' otteniamo la risposta a y.

TEOREMA

Sia y un quadrato in $\mathbb{Z}_p^*$ e sia $r \in_R [0, \dots, \frac{p-1}{2}]$ (esponente a caso per una possibile radice quadrata di un numero, presente nella prima metà)
Sia $y' = y \cdot g^{2r}$, 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 $z \cdot g^r$ è PSQR di $y \cdot g^{2r} \Leftrightarrow z$ è PSQR di y

$$\text{Sappiamo che } \sqrt{g^{2x} g^{2y}} = \begin{cases} g^x g^r \\ g^{x+r} g^{\frac{p-1}{2}} \end{cases}$$

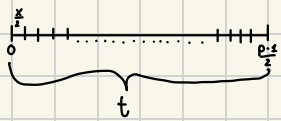
Con l'ipotesi $2x+2r < p-1$ allora g^{x+r} è la radice quadrata principale

Questo perché moltiplicando $y \cdot g^{2r}$ abbiamo un'operazione aritmetica poiché gli esponenti sono sommati e non si ricade nella radice quadrata successiva, ovvero quella dopo la prima metà (l'esponente di y è $2x$)
Moltiplicare quindi fa sì che non si vada oltre $p-1$, quindi il logaritmo discreto del prodotto è la somma dei logaritmi discreti; andar oltre $p-1$ significa tornare 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 $\frac{x}{2}$ sia nel primo intervallo, la probabilità di trovare un r dove $\frac{x}{2}+r$ non vada oltre $\frac{p-1}{2}$, ovvero $\frac{t-1}{t}$. La probabilità che la risposta sia corretta è la combinazione della probabilità di soddisfacibilità del teorema, ovvero $\left(\frac{t-1}{t}\right)$ e la probabilità della risposta al problema trasformato, ovvero $\left(\frac{1}{2}+k^{-c}\right)$

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 qual 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

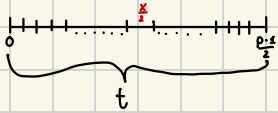
$$\frac{t-1}{t} \geq \frac{1}{2}+k^{-c} \Longrightarrow t \geq \frac{2}{1+2k^{-c}}$$

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 $i\,\frac{p-1}{2t} \leq \frac{x}{2} < (i+1)\frac{p-1}{2t}$

$$g^{x'} = y \cdot g^{\frac{p-1}{t} i} \Longrightarrow g^{x'} = g^{x} \cdot g^{-\frac{p-1}{t} i} \Longrightarrow x' = x - \frac{p-1}{t} i$$

Quindi x' sta 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 $g^{\frac{p-1}{t} i}$

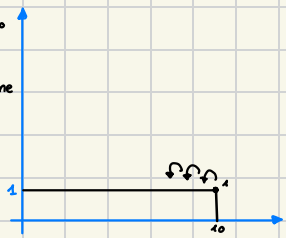
Non conoscendo l'intervallo in cui si trova $\frac{x}{2}$, possiamo provare a risolvere il problema per tutti gli intervalli. Essendo quantità polinomiali in K.

→

Lezione 12

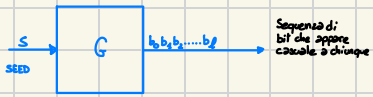
Vogliamo costruire una sequenza di bit che sia equivalente ad un lancio di moneta. Se volessimo utilizzare all'interno di un programma dei bit che sono casuali, avremmo bisogno di un processo di lancio di moneta all'interno del nostro calcolatore. All'interno del processo di lancio di moneta utilizzato 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 deterministiche. Il processo non è casuale, ma deterministico; ma appare come processo casuale agli occhi di chi lo osserva.

$f(i) = 1$ SSE IN ESPANSIONE π ESISTE UNA SEQUENZA DI ALMENO i 5



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 nessuno con potenza di calcolo probabilistico polinomiale sia in grado di capirne la distribuzione. Chiunque non si deve accorgere del fatto che non stiamo buttando con sequenze casuali, ma con sequenze pseudocasuali.

Generazione Sequenze Bit Pseudocasuali PRSG



Il seme s viene scelto in maniera realmente casuale ed è corto ed ha K bit e l'output ha l bit, normalmente con $l > K$. Il generatore 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, nonostante ci sia.

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

① $\forall_A \forall_c \exists_K \forall_{K > K} \forall_{\ell_{poly}} \left| P_r[s \in_R \{0,1\}^K;\ b_1, b_2, b_3 \dots b_\ell, b_{\ell+1} = G(s);\ b = A(b_0 \dots b_\ell);\ b == b_{\ell+1}] - \frac{1}{2} \right| < K^{-c} = K^{-\omega(1)}$

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 ovvero che abbia riconosciuto la sequenza pseudocasuale come tale:

$$P_K^{D,G} = P_r[s \in_R \{0,1\}^K;\ b_0 b_1 \dots b_\ell \leftarrow G(s);\ D(b_0 b_1 \dots b_\ell) = 1]$$

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

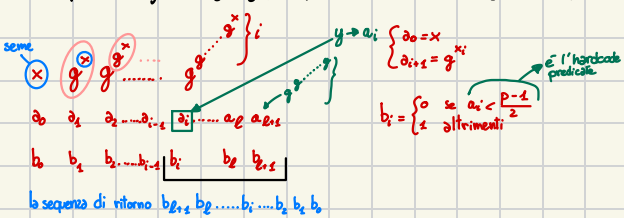
$$P_K^{D,R} = P_r[b_0 b_1 \dots b_\ell \in_R \{0,1\}^{\ell+1};\ D(b_0 b_1 \dots b_\ell) = 1]$$

Quindi la probabilità che il distinguisher restituisca 1 nei due casi differisce di una quantità più piccola di qualsiasi polinomio

② $\forall_{D \in PPT} \left| P_K^{D,G} - P_K^{D,R} \right| < K^{-\omega(1)}$

① e ② sono equivalenti, ciò 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 $x \in_R \{0, \dots, p-1\}$ e definiamo:



$$\begin{cases} a_0 = x \\ a_{i+1} = g^{x_i} \end{cases}$$

$$b_i = \begin{cases} 0 & \text{se } a_i < \frac{p-1}{2} \\ 1 & \text{altrimenti} \end{cases}$$

è l'hardcore predicate

Utilizzando la definizione 3.8, supponiamo per assurdo che esiste un algoritmo in grado di predire il bit successivo con un vantaggio più grande di qualche polinomio. I bit presi in input sono $b_\ell, \dots, b_2$, quindi l'algoritmo A predice il bit b_1. Ma a_3 è l'hardcore predicate di b_2, quindi non è possibile predire il bit b_2 con un vantaggio maggiore di qualche polinomio

$a_3 = g^{g^{g^x}}$ e $a_2 = g^{g^x}$

La costruzione della sequenza $a_0, a_1, a_2, a_3, \dots, a_\ell$ e della sequenza $b_\ell, b_{\ell-1}, b_{\ell-2}, b_{\ell-3}, \dots, b_0$ è facilmente costruibile. Se i bit fossero stati restituiti in ordine sequenziale $b_0, b_1, b_2, b_3, \dots, b_\ell$ 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 funziona 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 ragione non sono in grado di calcolarlo conoscendo i bit, ma tale dimostrazione non mi permette di catturare tale cancello.

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 $b_\ell, \dots, b_0$ soddisfa la definizione 3.9 allora soddisfa la definizione 3.8. Quindi $b_\ell, \dots, b_0$ é indistinguibile da una sequenza casuale e possiamo concludere che $b_0, \dots, b_\ell$ é indistinguibile da una sequenza casuale

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

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

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

Distinguere equivale a predere

TEOREMA

1. Sia distinguere equivalente a

$$\forall_{D \in PPT} \left| P_K^{D,G} - P_K^{D,R} \right| < K^{-\omega(1)}$$

2. Sia prevedere equivalente a:

$$\left| \mathbb{P}[\, s \in_R \{0,1\}^k ;\ b_0 b_1 \dots\dots b_\ell \leftarrow A(b_0 b_1 \dots\dots b_\ell);\ b = b_{\ell+1}] - \frac{1}{2} \right| \leq K^{-\omega(1)}$$

Allora:

- $1 \Rightarrow 2$ e $2 \Rightarrow 1$

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

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

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

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

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

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

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

Mentre la probabilità che A con input $b_0, \dots\dots, b_{\ell-1}$ indovini b_ℓ é 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'attaccante per la proprietà (1), allora violiamo la proprietà (2)

($2 \Rightarrow 1$) Disponiamo di una sequenza di bit $b_1, \dots\dots, b_\ell$ generata da G e una sequenza di bit $r_1, \dots\dots, r_\ell$ generata in maniera casuale e indipendente da G, supponiamo che D sia un distinguisher per $b_1, \dots, b_\ell$ e $r_1, \dots, r_\ell$

Sappiamo che:

$$S_0 = b^1 \dots\dots b^{i-1} b^i b^{i+1} b^{i+2} \dots b^\ell$$

$$S_1 = r^1 \dots\dots r^{i-1} r^i r^{i+1} r^{i+2} \dots\dots r^\ell$$

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

$$\left| P_K^{D,G} - P_K^{D,R} \right| \geq \frac{1}{2} - K^{-c}$$

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

Quindi

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

$$S_i = b^1 \dots b^{i-1} b^i r^{i+1} r^{i+2} \dots r^\ell$$

$$S_{i+1} = b^1 \dots b^{i-1} b^i b^{i+1} r^{i+2} \dots r^\ell$$

In qualche modo il distinguisher si comporta diversamente quando la sequenza differiscono di un solo bit.
Senza perdita di generalità $P_{i+1} - P_i \geq k^{-c}$, supponiamo che la probabilità che D restituisca 0 su input $b_1, \dots, b_i, \overline{b_{i+1}}, \dots, r_\ell$, sia x, allora la probabilità che A indovini il bit successivo è:

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

$$\frac{1}{2} \cdot P_{i+1} + \frac{1}{2} \cdot x$$

Proviamo attualmente a cercare una relazione linearmente indipendente con P_{i+1}. Il valore equivale alla probabilità che il distinguisher restituisca 1 su input $b_1,\dots,b_i,r_{i+1},\dots,r_\ell$

$$P_i = P[\text{il bit i-esimo} = b_{i+1}] \cdot P[D \text{ dia 1 su } b_1,\dots,b_i,b_{i+1},\dots,r_\ell] + P[\text{il bit i-esimo} = \overline{b_{i+1}}] \cdot P[D \text{ dia 0 su } b_1,\dots,b_i,b_{i+1},\dots,r_\ell] =$$

$$= \frac{1}{2} \cdot P_{i+1} + \frac{1}{2} \cdot (1-x) = \frac{1}{2} \cdot P_{i+1} + \frac{1}{2} - \frac{1}{2} \cdot x$$

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 Z_n^*
La funzione di elevamento al quadrato è una funzione oneway trapdoor in 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:

$$LSB(x) = \sqrt{x}$$ un hardcore predicate per l'elevamento al quadrato è 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 $p,q \equiv 3 \bmod 4$) allora per ogni quadrato x esiste una sola radice quadrata y tale che $y \equiv \sqrt{x} \bmod n$
Sia $n = pq$ con p e q primi di Blum scelti casualmente, sia $x \in_R Z_n^*$, allora:

Z_n^* $n = pq$

$x \quad x^2 \quad (x^2)^2 \quad ((x^2)^2)^2 \dots\dots x^{(2^\ell)}$

$a_1 \quad a_2 \quad a_3 \quad a_\ell$

HCP(a_1) HCP(a_2)....

LSB(a_1) LSB(a_2)....

$$\begin{cases} x_1 = x^2 \\ x_{i+1} = x_i^2 \end{cases}$$

Definiamo quindi $b_i = LSB(a_i)$ quindi:

$a_1 \quad a_2 \quad a_3 \quad \dots\dots a_\ell$

$x^2 \quad (x^2)^2 \quad ((x^2)^2)^2 \quad x^{2^\ell}$

$b_1 \quad b_2 \quad b_3 \quad b_\ell$

Dove il valore di ritorno sarà la sequenza $b_\ell, b_{\ell-1}, \dots, b_1$. Possiamo ovviamente restituire i bit anche in enumerazione crescente, ovvero $b_1, b_2, b_3 \dots b_\ell$.

La differenza è che se n è noto a tutti allora è possibile restituire i bit solo nell'ordine $b_1, b_2, \dots, b_\ell$.

Blum Blum Shub

$p,q \equiv 3 \mod 4$

Sia $D \in$ PPT un distinguisher

$P_k^{D,U}$ $\quad Pr[D$ mett 1 su input ℓ bit casuali$]$

$P_k^{D,G}$ $\quad Pr[D$ mett 1 su input ℓ bit generati$]$

$Pr[s \in_R \{0,1\}^\ell ; D(s)==1]$

$Pr[s \in_R \{0,1\}^k ; G()]$

$\forall D \in PPT \quad |P_k^{D,U} - P_k^{D,G}| < k^{-\omega(1)}$

Lezione 13

$$\left| Pr[s \in_R \{0,1\}^k ; b_1 \ldots\ldots b_\ell b_{\ell+1} = G(s) ; b = A(b_1, \ldots\ldots, b_\ell) ; b == b_{\ell+1}] - \frac{1}{2} \right| < K^{-c}$$

$$|P_K^{D,U} - P_K^{D,G}| < K^{-c} \qquad |P_i - P_{i+1}| > K^{-c}$$

$P_i \quad G_1 \ldots\ldots G_i \; R_{i+1} R_{i+2} \ldots\ldots$

$P_{i+1} \quad G_1 \ldots\ldots G_i \; G_{i+1} R_{i+2} \ldots\ldots$

ℓ bit $\quad S(i): \; G_1 G_2 \ldots\ldots G_i R_{i+1} \ldots\ldots R_\ell$

$S(0)$: TUTTI CASUALI

$S(\ell)$: TUTTI GENERATI

$$P(i) = P_K^{D,S(i)}$$

$$|P(\ell) - P(0)| > K^{-c}$$

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

$D(b_1 \ldots\ldots b_i R_{i+1} \ldots\ldots R_\ell)$

$$A(b_1 \ldots\ldots b_i) = \begin{cases} R_i & \text{se } D=1 \\ \overline{R_i} & \text{se } D=0 \end{cases}$$

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

$$Pr[A(b_1 \ldots\ldots b_i) = b_{i+1}] = Pr[R_{i+1} = b_{i+1}] \quad P_{i+1}$$
$$+ Pr[\overline{R_{i+1}} = b_{i+1}]$$

Sia x $\quad Pr$ $D=1$ se input $\quad b_1 \ldots\ldots b_i \; \overline{b_{i+1}} \; R_{i+2} \ldots\ldots$

$$Pr[A(b_1 \ldots\ldots b_i) = b_{i+1}] = \frac{1}{2} P_{i+1} + \frac{1}{2} - \frac{1}{2} x$$
$$= \frac{1}{2} + \frac{1}{2}(P_{i+1} - x) \longrightarrow \frac{1}{2} + \frac{1}{2}(P_{i+2} - 2P_i + P_{i+1}) = \frac{1}{2} + (P_{i+1} - P_i)$$

$$\underline{P_i} = Pr[R_{i+1} = b_{i+1}] \cdot P_{i+1} = \underline{\frac{1}{2} P_{i+1} + \frac{1}{2} x}$$
$$+ Pr[\overline{R_{i+1}} = b_{i+1}] \times$$

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

Blum Goldwasser

P	n
S	p,q

$n = pq \quad p,q$ primi di blum

$E(m) \quad m = b_1 \ldots b_\ell$

$$x \in_R Z_n^* \longrightarrow \boxed{G_{BBS}} \longrightarrow k_1 \ldots\ldots k_\ell$$
$$b_1 \ldots\ldots b_\ell \; \oplus$$
$$c_1 \ldots\ldots c_\ell \qquad x^{(2^{\ell+1})}$$

D

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

$m, f(m)$

$m' \quad f(m')$

Enumeriamo le funzioni e condividiamo l'indice

$$|A \to B| = |B|^{|A|}$$

→ calcolo combinatorio

$|A| \cdot |A| \cdot |A| \ldots\ldots$ → $|A|^n$

n

$z \to a_3 \in A$

$y \to a_2 \in A$

$x \to a_1 \in A$

$|A|$ possibilità

$|A|$ possibilità

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

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

→ serve una quantità esponenziale di bit

$F_k \subseteq U_k$

1. $|F_k| = 2^k$
2. Esiste algoritmo PPT che calcola

   $(i, x) \longmapsto f_i(x)$
3. ....

Lezione 14

$A \to B$
funzione che mappa un elemento dell'insieme A in un elemento dell'insieme B

$A \mapsto B$
Dice come opera la funzione, ovvero come vengono utilizzati i valori in A

$U_K : \{0,1\}^K \to \{0,1\}^K \qquad K2^K$

- $F_K \subseteq U_K \qquad |F_K| = 2^K$
- (La funzione $(i,x) \mapsto f_i(x)$) $\in$ PPT
- Sia $D \in$ PPT un algoritmo che interroga f quando vuole

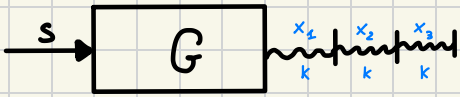
  Sia $P_K^{D,U}$ Prob D rest 1 se $f \in_R U_K$

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

$$\left| P_K^{D,U} - P_K^{D,F} \right| < K^{-\omega(1)}$$

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

G, S



$f(x_1)$

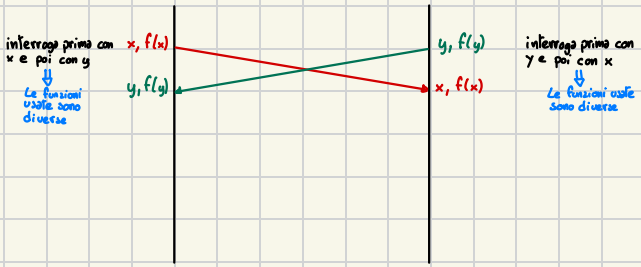
$f(x_2)$

$f(x_3)$

$f(x_2)$

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

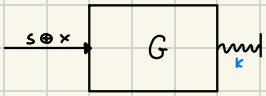


$f_s(x)$ = x-esimo gruppo di K bit generato da $G(s)$

$f_s(1^K)$
la sequenza di K 1 ovvero $2^K - 1$

non riusciamo nemmeno a calcolarla in tempo polinomiale
$K(2^K - 1)$

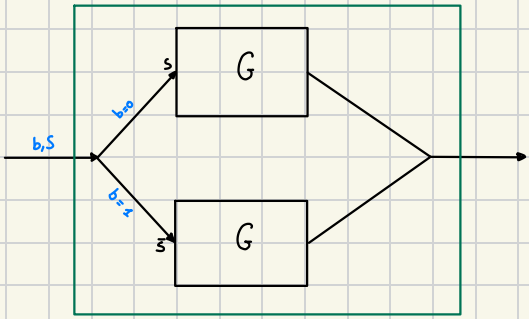
$G, \underline{S}$



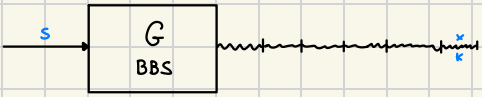
$f_s(x)$
$f_s(y)$ → Crittoanalisi differenziale

per ogni x $f_s(x) = f_s(\bar{x})$

generatore da 100x bit



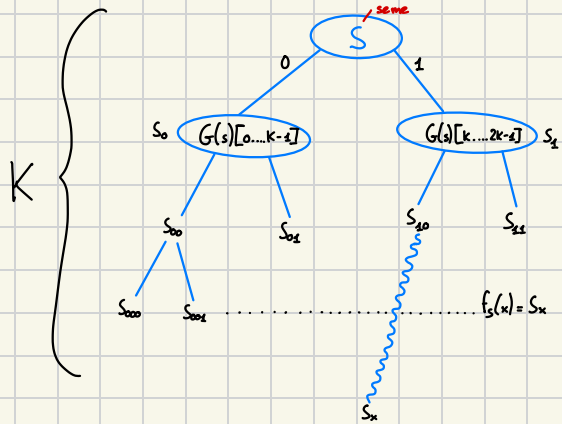
$f_s(0) = f_s(1^k)$

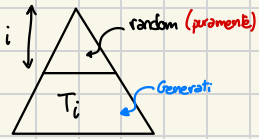


$f_s(x)$ = x-esimo seq.

$\underline{K_x}$

$s^{\varphi(n)} = 1$ $\qquad s^{\overbrace{(2^x)\bmod \varphi(n)}^{y}} = s^{y + k\varphi(n)}$





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

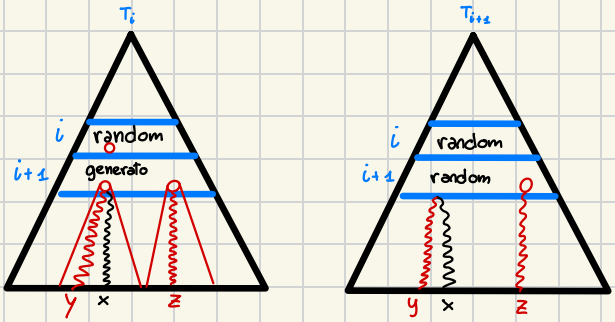
Il seme è random, il resto è generato

T_0 è la funzione pseudocasuale

T_k è la funzione casuale

$P_i \triangleq P_k^{D,T_i}$ $\quad |P_0 - P_k| > k^{-c}$

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



Sto facendo più esperimenti indipendenti

Forger $\quad m_1 m_2 \ldots m_\ell$

$(m, \underline{\underline{f(m)}})$

Lezione 15

Funzione pseudocasuale f_s

Autenticazione messaggi $(m, f_s(m))$ $\qquad P_r[F \text{ successo}] > k^{-c}$

Forger $\in$ PPT che interroga f_s un numero arbitrario di volte

$$(m_1, f(m_1))\ (m_2, f(m_2)) \ldots\ldots (m_\ell, f_s(m_\ell))$$

PRODUCE UNA COPIA (m, σ)

F ha successo se $m \neq m_i \quad \forall i \in \{1 \ldots\ldots \ell\}$

$$\sigma = f_s(m)$$

D: USA IL FORGER PER PRODURRE (m, σ)

Verifica se $\sigma = f(m)$. In tal caso output 1

$$\boxed{\begin{array}{l} P_k^{D,G} > k^{-c} \\ P_k^{D,U} = \frac{1}{2^k} \end{array}}$$

$M = m_1 m_2 \ldots\ldots m_\ell$

$F(m) \triangleq f(m_1) \oplus f(m_2) \oplus \ldots\ldots\ldots\ldots \oplus f(m_\ell)$

$$f(mm) = f(m) \oplus f(m) = 0$$

$m_1\ m_2\ m_3,\ F(m_1, m_3, m_3)$

$m_1\ m_3\ m_2,$

$F(m) \triangleq f(m_1) \oplus f(m_2) \oplus \ldots\ldots\ldots\ldots \oplus f(m_\ell)$

$\quad = f(<1>m_1) \oplus f(<2>m_2) \oplus \ldots\ldots \oplus f(<\ell>, m_\ell)$

$F(mm) = f(<1>m) \oplus f(<2>m)$

$F(m_1 m_2 m_3) = f(<1>m_1) \oplus f(<2>m_2) \oplus f(<3>m_3)$

$F(m_1 m_3 m_2) = f(<1>m_1) \oplus f(<2>m_3) \oplus f(<3>m_2)$

$F(m_1 m_2) = \cancel{f(<1>m_1)} \oplus \cancel{f(<2>m_2)}$

$F(m_1' m_2) = f(<1>m_1') \oplus \cancel{f(<2>m_2)}$

$F(m_1, m_2') = \cancel{f(<1>m_1)} \oplus f(<2>m_2')$

$\oplus$

$F(m_1', m_2') = f(<1>m_1) \oplus f_2(<2>m_2')$

$F(m_1 \ldots\ldots m_\ell) = f(m_\ell \oplus \ldots\ldots\ldots f(m_3 \oplus f(m_2 \oplus f(m_1))))$

$F_i(m_1 \ldots\ldots m_\ell) = \underbrace{\qquad\qquad}_{f \text{ pseudocasuale}} \underbrace{\qquad\qquad}_{i \text{ usa } f \text{ casuale}}$

$F_i \quad F_{i+1}$ il distinguisher ne vede la differenza perché è maggiore di un polinomio

Lezione 16

Un sistema di firma digitale è l'insieme di 3 algoritmi

generate key $\leftarrow G: 1^k \to (S_k, P_k)$ (secret, public)

sign $\leftarrow S: m, S_k \mapsto \sigma$ → stringa di firma → in firma asimmetrica solo chi la può generare

verify $\leftarrow V: m, \sigma, P_k \to \{0,1\}$

La probabilità di verificare una firma corretta deve essere 1

$$P_r[V(m, S(m, S_k), P_k) = 1] = 1$$

Sia F PPT che chiede firma di $m_1 \ldots\ldots m_\ell$ e produce una copia (m, σ)
Diciamo che F ha successo se $m \notin \{m_1, \ldots\ldots, m_\ell\}$ e $V(m, \sigma, P_k) = 1$

$P_r[F \text{ ha successo}] < k^{-\omega(1)}$

In RSA abbiamo

(m, m^d)

1° forger

(m_1, σ_1) (m_2, σ_2)

(m_1, m_1^d) (m_2, m_2^d)

$(m_1 m_2, (m_1 m_2)^d)$

$(m_1 m_2)^d = m_1^d m_2^d$

2° forger (non interroga nemmeno il sistema)

(m^e, m)

Costo di firma estremamente costoso

Proviamo a usare una funzione

$$H: \{0,1\}^* \to \{0,1\}^k$$

Hash

Prendo una qualsiasi sequenza arbitraria di bit e la trasformo in una sequenza alla k bit

Firmo l'hash di un messaggio

$(m, H(m)^d)$

$(m, \sigma): \sigma^e \overset{?}{=} H(m)$

Non tutte le funzioni di hash vanno bene (la funzione NON deve essere omomorfica

Siano m_1, m_2 TC $H(m_1) = H(m_2)$ ma è difficile in caso funzioni di hash che evitano collisioni

MD MD2 ..... MD5

SHA SHA1 .... SHA2

Bitcoin si basa sulla difficoltà di trovare collisioni

Blind Signature

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

A B

$m, \quad b \in_R Z_n^*$

elemento casuale di Z_n^* perché b è casuale

$mb^e \longrightarrow (mb^e)^d = m^d b$

$m^d \leftarrow \frac{m^d b}{b} \longleftarrow m^d b$

Con le opportune sanzioni questo sistema va più che bene

Supponiamo che H non sia one-way

H

m $\quad H(m) \rightarrow \boxed{A} \xrightarrow{m'} H(m') = H(m)$

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 resistence. Assurdo. Quindi H è one way

Lezione 17

Bit commitment

$B \xrightarrow{R} A$

$\xleftarrow{E_K(b,R)}$

$\xleftarrow{K}$

$E_{K_1}(0,R) = E_{K_2}(1,R)$

$A \xrightarrow{H(b,R_1,R_2),\, R_1} B$

$\xrightarrow{b, R_2}$

$Bob \xrightarrow{R} A$

A genera S per PRSG G

Usa G per generare $x_1, x_2, \ldots, x_\ell$ seq di bit

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

Apertura busta, Alice invia a Bob il seme e lui genera la sequenza casuale

Schema barriera (n,n)

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

$x_1, x_2, \ldots, x_{n-1} \in_R \{0,1\}^\ell$

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

Solo quando ho tutte le parti riesco a leggere il segreto

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

Voglio costruire uno schema (n, K)

Se voglio costruire un messaggio che si decifri con K agenti su n

Ma allora costruisco un polinomio $P(x)$ di grado $K-1$ e distribuisco i valori su esso

- retta si ottiene con 2 punti
- parabola con 3 punti
- pol grado n con $n+1$ punti

- Devo costruire $P(x)$ di grado $K-1$ t.c. $P(0)=S$

  P deve essere casuale

- Ad ogni agente $i \in \{1, \dots, n\}$ consegnare $(i, P(i))$

$$\underset{\underset{S}{\parallel}}{a_0} + a_1 x + a_2 x^2 + \dots + a_{K-1} x^{K-1}$$

I crittografi mangiatori

Ci sono 3 crittografi che vanno a cena

A un tavolo a fianco c'è un tipo sospetto

Arriva il conto ma è già stato pagato

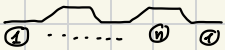
I crittografi vogliono sapere se a pagare è stato uno di loro

Il bit generato dal lancio di moneta viene condiviso solo con il crittografo di destra

Noi vogliamo calcolare

$f(x_1, x_2, x_3) = x$ senza sapere nulla su x_1, x_2, x_3

$$c_i = \begin{cases} 0 & b_i = b_{i-1} \\ 1 & \text{altrimenti} \end{cases}$$



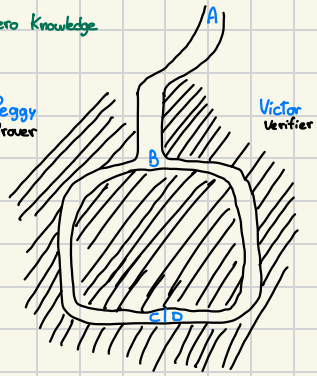
alla fine ritorno al punto 1

incontro un numero pari di cambi

I numeri di bit a 1 annullati è pari

Lezione 18

Zero Knowledge



PV sono in A

P raggiunge B lancia moneta e sceglie C,D

V raggiunge B lancia moneta per chiedere C,D

P obbedisce

Interactive Proof System per un linguaggio L

P potenza illimitata

$V \in PPT$

- $\forall x \in L \quad P_r[(P,V) \text{ Accetta}] > \frac{2}{3}$ $\qquad p > \frac{1}{2} \qquad 1$
- $\forall x \notin L \ \forall P' \quad P_r[(P',V) \text{ Accetta}] < \frac{1}{3}$ $\qquad q < \frac{1}{2} \qquad \leq \frac{1}{2}$

IPS = PSPACE

Quadratic non residuocity

$\{(x,n) \mid x \text{ non quadrato in } Z_n^*\}$

$V \quad b_1 \ldots\ldots b_\ell \in_R \{0,1\}$

$\forall_i \; W_i = \begin{cases} y_i^2 & \text{se } b_i = 0, \; y_i \in_R Z_n^* \\ x y_i^2 & \text{se } b_i = 1 \end{cases} \xrightarrow{w_1, \ldots\ldots, w_\ell} P$

$c_i = \begin{cases} 0 & \text{se } w_i \text{ QUAD} \\ 1 & \text{Altrimenti} \end{cases}$

$V \xleftarrow{c_1 \ldots\ldots c_\ell}$

Graph-non-iso $= \{(G_1, G_2) : G_1 \simeq G_2\}$

INIETTIVO
SURIETTIVO
BIETTIVO

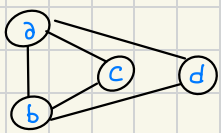
STUDIA

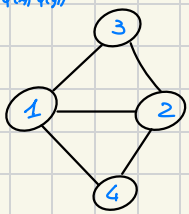
$f \; A \times A \longrightarrow A$

$f' \; B \times B \longrightarrow B$

$\varphi : A \to B$

$\varphi(f(x,y)) = f'(\varphi(x), \varphi(y))$





V sceglie a caso tra G_1, G_2 sia G

permuta G a caso in G'

$\xrightarrow{G'}$ P

$\xleftarrow{1 \text{ o } 2}$: GRAFO A CUI G' è isomorfo

View(x,h)

$M(x,h)$

Perfect zero Knowledge $\quad$ View$(x,h) = M(x,h)$

Statistical zero Knowledge $\quad \sum_{\alpha} \left| \text{VIEW}(x,h)[\alpha] - M(x,h)[\alpha] \right| < k^{-\omega(1)}$

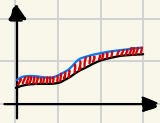
Computational Z_K $\quad$ VIEW(x,h), $M(x,h)$ POLY INDISTINGUIBILI

Lezione 19

View (x, h) (α) — hint

esiste algoritmo polinomiale

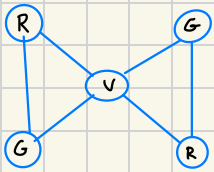
$M(x,h)$ (α)



- perfect zero-knowledge $\forall x \ \forall h$ $View(x,h) = M(x,h)$
- statistical zero-knowledge $\sum_{\alpha} | View(x,h)(\alpha) - M(x,h)(\alpha) < K^{-u(x)}$
- computational zero-knowledge $View(x,h)$, $M(x,h)$ sono computazionalmente indistinguibili

Grafo tricolorabile

$$\left(\frac{E-1}{E}\right)^n < \frac{1}{2}$$



Alla fine il prover facendo gli esperimenti ho in mano la tricolorazione → non è zero knowledge

Tra un esperimento e l'altro scambio completamente i colori, così il prover non ottiene informazioni in più

ad es $G \to V$
$V \to R$
$R \to G$

Io tricolaro il grafo e invio tramite bit commitment

....

Ciclo Hamiltoniano

G

P permuta G in G'

codifica matrice di adiacenza di G' in H

V

H →

← Domanda

Lancia moneta per chiedere

- ciclo su G'
- isomorfismo $G \approx G'$

P

Risposta 1 Apre solo le buste di un ciclo

2 Apre tutte le buste e invia una permutazione

Sequenziale

C_1 →

← Q_1

R_2 →

C_2 →

← Q_2

R_2 →

⋮

C_K →

← Q_K

R_K →

Parallela

$C_1 C_2 \ldots\ldots C_K$ →

← $Q_1 Q_2 \ldots\ldots Q_K$ ← $H(C_1 \ldots\ldots C_K)$

$R_1 R_2 \ldots\ldots R_K$ →

Non interactive zero knowledge