Appunti Crittografia

Alessandro Pagiaro

3 gennaio 2015

Indice

1 La classi di problemi

1.1 La classe co-NP

È semplice certificare l'esistenza di una soluzione. Più difficile è certificare la non-esistenza di una soluzione.
La classe co-NP è la classe dei problemi i cui complementari sono in NP.

$$PsubseteqNP \quad P \subseteq co-NPP \subseteq NP \cap co-NP$$

La vera domanda è se

$$NP = co-NP$$

Si crede che $P \neq NP$.

Definizione 1. *Un problema si dice NP-hard se*

$$\forall B \in NP, B \leq_p A$$

(dove il simbolo $\leq_p$ vuol dire che si può ridurre polinomialmente)

2 Introduzione al concetto di probabilità

X : 01101011100011001101
Y : 11111111111111111111
Z : 10011101101010110100

Noi possiamo dire, secondo la classica definizione di probabilità, che

$$P(X) = P(Y) = P(Z) = \left(\frac{1}{2}\right)^{20}$$

3 Casualità di Kolmogorev, 1960

Sia:
A_y := Algoritmo che genera la sequenza y di lunghezza n

Tuttavia la sequenza Y, vista al caso precedente, avremmo potuto generarla con un algoritmo del tipo:
$A_y =<$ genera n uni $>$
Chiediamoci qual'è la dimensione dell'algoritmo:

$$|A| = \text{dim in bit dell'algoritmo} = cost + \Theta(logn)$$

Qual è la cardinalità di Y?

$$|Y| = n \gg |A_y|$$

Definizione 2. *Diremo che una sequenza è casuale se $|A|$ è tanto quanto la lunghezza del dato casuale genarato. Cioè non ammetto un algoritmo di generazione la cui rappresentazione binaria sia più corta della stringa stessa.*

Sempre riferendoci all'esempio precedente, l'algoritmo che genera X poteva essere solo della forma

$$A_x =< \text{contiene X al suo interno e lo mette nel risultato} >$$

A_x ha la cardinalità:

$$|A_x| = cost + |X| = cost + \Theta(|X|)$$

4 Complessità di Kolmogorov di h in S_i

Siano $S_1, S_2, ..., S_i, ...$ un'infinità numerabile di modelli di calcolo. Indicheremo con $S_i(p) = h$ l'esecuzione del programma p sul modello S_i che restituisce come output h.

La lunghezza del programma più corto che genera h su S_i si può definire nel seguente modo:

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

Se h_i non presenta alcun tipo di regolarità allora K_{S_i} della sequenza h (cioè $K_{S_i}(h)$) sarà

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

poichè l'unico modo affinchè restituisca h è che lo contenga al suo interno.

Supponiamo che esista S_u, un sistema universale in grado di simulare ogni modello di calcolo ragionevole.

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

$q =< i, p >$ è un programma che su S_u genera h.
$|q| = |i| + |p| = |p| + \Theta(logi) = |p| + c \cdot i$

$$\forall i \; K_{S_u}(h) \leq K_{S_i}(h) + c \cdot i$$

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

Definizione 3. *h è casuale se* $K(h) \geq |h| - \lceil log(|h|) \rceil$

4.1 Conteggio delle sequenze casuali

$\forall n$

$$S = \#\text{sequenza lunghezza n} = 2^n$$

$$T = \#\text{sequenze non casuali di n bit}$$

Vogliamo ora dimostrare che $T < S$.

Dimostrazione.

$$N = \#\{\text{sequenze di lunghezza} < n - \lceil log_2 n \rceil\} = \sum_{i=0}^{n-\lceil logn \rceil - 1} 2^i = 2^{n-\lceil log_2 n \rceil} - 1$$

Notare che N contiene anche i programma "brevi" che generano le T sequenze non casuali di n bit.

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

Cioè

$$T < S$$

□

Possiamo dire che esistono certamente quindi sequenze casuali di n bit, qualunque sia n.

Studiamo ora $\frac{T}{S}$ per vedere che $T \ll S$.

$$\frac{T}{S} = \frac{2^{n-\lceil logn \rceil} - 1}{2^n} < \frac{2^{n-\lceil logn \rceil}}{2^n} < \frac{1}{2^{\lceil logn \rceil}}$$

Che altro non vuol dire che $\frac{T}{S} \to 0$ per $n \to \infty$. Quindi ci sono più sequenze casuali che non-casuali.
E qui sorge un piccolo problema:

Teorema 1. *Stabilire se una sequenza arbitraria è casuale secondo Kolmogorov è un problema indicidibile.*

Dimostrazione. $\exists$ un programma $RANDOM$ t.c.

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

Definiamo ora
PARADOSSO()
for binary h = 1 to $+\infty$ do
{ if ($|h| - \lceil logh \rceil > |p|$ && $RANDOM(h) == 1$)
return h
}

PARADOSSO restituisce una sequenza casuale di lunghezza $|P|+1$ (almeno un $\exists$ sicuramente).
p: è la sequenza binaria rappresentata da PARADOSSO + RANDOM
$|p|$ = lunghezza del programma PARADOSSO + RANDOM. (Notare che non dipende da h, in quanto h comparirà solo come nome di variabile)

Quindi per:

$$|h| - \lceil logh \rceil > |p|\$$$

h non può essere casuale ($|p|$ è breve!). Inoltre per

$$RANDOM(h) == 1$$

h è casuale. Assurdo. L'algoritmo RANDOM non può esistere. □

5 Generazione di sequenza casuali

Sorgente: Genera una sequenza di bit $P(0) = P(1) = \frac{1}{2}$. La generazione di un bit è indipendente dalla generazione dei bit precedenti.

5.1 Generatori di numeri pseudo-casuali

Input : seme, sequenza breve di bit casuali
Output : sequenza arbitrariamente lunga che contiene una sottosequenza detta periodo che si ripete. Il periodo deve risultare casuale.
Seme: s bit $\rightarrow 2^s$ possibilità
Periodo: m bit $\rightarrow s^m$ possibilità
Deve risultare che $2^s \ll 2^m$.

Esempio. Generatore Lineare.

$$x_i = (ax_{i-1} + b) \text{ mod m}$$

con a,b,m interi positivi, x_0 seme, genera

$$x_0, x_1, x_2, ...$$

Il problema in questo caso è che se si genera un duplicato ($x_i = x_j$) la sequenza si ripete ($x_{i+1} = x_{j+1}$). Affinché venga garantito un periodo lungo m bit devono quindi valere queste tre condizioni:

$$\begin{cases} MCD(b, m) = 1 \\ (a-1)\text{deve essere divisibile da ogni fattore primo di m} \\ (a-1)\text{deve essere multiplo di 4 se anche m lo è} \end{cases}$$

Alcuni parametri consigliati, usati da molte librerie, sono:

$$\begin{cases} a = 3141592653 \leftarrow \pi \\ b = 2718281829 \leftarrow e \\ m = 2^{32} \\ seme = 0 \end{cases}$$

5.2 Generatore Polinomiale

$$x_i = (a_1 x_{i-1}^t + a_2 x_{i-1}^{t-1} + a_3 x_{i-1}^{t-2} + ... + a_t x_{i-1} + a_{t-1})$$

Si può facilmente verificare che questo generatore presenta gli stessi problemi del generatore lineare sopra illustrato.

5.3 Generatori Crittograficamente Sicuri

Definizione 4. *Un generatore si dice sicuro se le sequenze generate superano il test di prossimo bit, cioè non è possibile fare previsioni sugli elementi non ancora generati.*

Definizione 5. *Si definisce funzione one-way una funzione che dato x è "facile" calcolare $y = f(x)$. Dato y è "difficile" trovare $\{x|y = f(x)\}$.*

Proviamo a vedere il generatore lineare supera il test di prossimo bit.
Scelgo x_0 seme. Calcolo $f(x_0) = x_1$. Procedo calcolando $f(f(x_0)) = x_2$....

$$x_0, f(x_0), f(f(x_0)), ...$$

Calcolare $x_i \to x_{i+1}$ è facile, cioè sono tutte operazioni polinomiali.
Calcolare $x_{i+1} \to x_i$ è difficile.
Posso quindi usare tutte le chiavi in senso inverso così che uso solo x_{i+1} e successivamente userò x_i, affidandomi al fatto che $x_{i+1} \to x_i$ è un'operazione esponenziale.

5.4 Generatori binari

Per creare dei generatori binari partendo dai generatori di interi si usano i predicati "hard core" per una funzione one-way. Il predicato è facile da calcolare se si conosce x, difficile se si conosce l'immagine $y = f(x)$.

5.4.1 Test di prossimo bit

Un generatore binario supera il test di prossimo bit se $\nexists$ un algoritmo polinomiale in grado di produrre l'$(i+1) - esimo$ bit della sequenza pseudo-causale generata a partire dalla conoscenza degli $i - bit$ precedenti con probabilità significativamente maggiore di $\frac{1}{2}$.

5.4.2 Generatore BBS

Un generatore BBS (Blum, Blum, Shab) usa la seguente funzione:

$$f(x) = x^2 \text{mod n, con n non primo}$$

trovare infatti la radice di un numero modulo n è un problema difficile.
Predicato hard-core per generare numeri binari:
$b(x)$ = "x è dispari"

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

Criteri di scelta dei parametri Affinché il generatore BBS funzioni correttamente occorre che il parametro n venga scelto con i seguenti criteri:

$$n = pq$$

p, q primi grandi t.c.

$$p \bmod 4 = 3$$

$$q \bmod 4 = 3$$

$$2\left\lfloor \frac{p}{4} \right\rfloor + 1, 2\left\lfloor \frac{q}{4} \right\rfloor + 1 \text{ devono essere primi}$$

Il fatto che p e q siano congruenti a 3 modulo 4 garantisce che il
$MCD(\phi(p-1), \phi(q-1))$ sia piccolo (il che rende più lungo il ciclo per cui x_i torna ad essere uguale a x_0)

Problemi del generatore Il problema principale di questo generatore è la latenza, è computazionalmente dispendioso infatti elevare al quadrato numeri molto grandi.

5.5 Generatore consigliato

Siano:
C(m, k) una funzione di cifratura di cifrario simmetrico
k: chiave segreta
r = # bit del blocco usato nel cifrario (es. $r = 64$ nel cifrario DES)
s = seme lungo r
m = # blocchi da r bit da generare

GENERATORE(s, m):
d = rappresentazione su r bit di data e ora attuale
y = C(d,k)
z = s
for(i=1; i $\leq$ m; i++) {
$x_i = C(y \oplus z, k)$;
$z = C(y \oplus x_i, k)$;
comunico x_i;
}

6 Algoritmi Randomizzati

Esistono due tipi di algoritmi randomizzati:

Algoritmi di tipo Las Vegas: forniscono un risultato sicuramente corretto, tempo probabilmente breve

Algoritmi di tipo Montecarlo: forniscono un risultato probabilmente corretto in tempo sicuramente breve, probabilità di errore arbitrariamente piccola e matematicamente misurabile.

6.1 Test di primalità (Miller, Rabin)

Questo algoritmo presenta le caratteristiche tipiche di un algoritmo probabilistico di tipo Montecarlo. Permette di decidere in tempo polinomiale se un numero N è primo con probabilità di errore di 2^{-k} con k grande a piacere.
Sia quindi:
N un numero dispari (per i numeri pari è facilmente verificata la primalità)
n $= |N| = \Theta(logN)$
N-1 $= 2^w \cdot z$ (con z dispari e w,z calcolati in tempo polinomiale).
Se N è primo $\Rightarrow \forall y\ 2 \leq y \leq N-1$

$$P1)\ MCD(N, y) = 1$$

$$P2)\ (y^z \bmod \mathrm{N} = 1) \vee (\exists 0 \leq i \leq w-1 \text{ t.c. } (y^{2^i \cdot z} \bmod \mathrm{N} = -1))$$

La seconda proprietà non è sempre vera ma i numeri composti che le soddisfano entrambe sono pochi.

Lemma di Miller, Rabin Se N è composto, il numero di interi $y \in [2, N-1]$ che soddisfano 1) e 2) è minore di $\frac{N}{4}$. In formule:

$$\#\{y \in [2, N-1] | P1(y) = TRUE \wedge P2(y) = TRUE\} < \frac{N}{4}$$

Preso a caso $y \in [2, N-1]$ possiamo dire che se

$(P1(y) = FALSE) \vee (P2(y) = FALSE) \Rightarrow$ N è certamente composto

Se invece

$$P1(y) \wedge P2(y) \Rightarrow \begin{cases} \text{N è primo con probabilità} > \frac{3}{4} \\ \text{N è composto con probabilità} < \frac{1}{4} \end{cases}$$

VERIFICA(N, y):
```
if(P1==false OR P2 == false)
return 1; //N è certamente composto
else return 0; //N è primo com probabilità > 3/4
```

TestMR(N, K):
```
for(i=1; ilek; i++){
scegli a caso y in [2, N-1];
if( VERIFICA(N,y) == 1) return 0; //m è certamente composto
}
return 1; //n è probabilmente primo con errore < 1/4^k
```

Complessità VERIFICA P1) richiedere di calcolare l'$MCD(N, y)$ (con l'algoritmo di Euclide) = $O(log^3 N)$
P2) Condizione 1:

$$y^2 \text{ mod N}$$

costa $O(z \cdot$costo moltiplicazione) con $1 \leq z \leq \frac{N-1}{2}$. Il numero di moltiplicazioni è $O(N)$, esponenziale in $n = |I| = \Theta(logN)$ Cerchiamo quindi un modo più efficiente per eseguire la potenza di un numero. (Verrà illustrato successivamente).
La complessità totale risulterà quindi:
P1) $O(log^3 N)$
P2) $z = O(N) \Rightarrow T(n) = O(n^3)$ —— DOPO ALTRI CALCOLI ——
Costo totale di TestMR = $O(k \cdot log^3 N)$

6.2 Esponenziazione Veloce

Vediamo qui il metodo delle quadrature successive.

$$x = y^z \text{ mod s}$$

dovey, z, s sono numeri interi dello stesso ordine di grandezza.

Passo 1 Si scrive z come somma di potenze di 2

$$z = 2^{p_1} + 2^{p_2} + 2^{p_3} + ... + 2^{p_t}$$

con $p_1 < p_2 < p_3 < ... < p_t$
$t = O(logz)$
$p_t = \Theta(logz)$.
es. $45 = 1 + 2 + 8 + 32$, $t = 4$, $p_t = 5$.

Passo 2 Calcolo (y^{2^s} mod s) con $1 \leq j \leq p_t$

$$y^z \text{ mod s} = (y^{2^{p_1}+2^{p_2}+...+2^{p_t}}) \text{ mod s} = (y^{2^{p_1}} \cdot y^{2^{p_2}} \cdot ... \cdot y^{2^{p_t}}) \text{ mod s}$$

Calcolare y^{2^j} mod s è semplice, infatti

$$y^{2^j} \text{ mod s} = (y^{2^j-1})^2 \text{ mod s}$$

E così ricorsivamente si può calcolare la potenza di un numero in O(log t) passi.

Passo 3 Calcolare $x = y^z$ mod s = $\prod_{e=1}^{t} y^{2^{p_e}}$ mod s

7 Generatore di Numberi Primi

Possiamo quindi ora spiegare come si genera un numero primo. Il concetto alla base dell'algoritmo riportato è quello di generare un numero casuale e verificare

con il test di primalità se questo è primo o no. Questo algoritmo è buono in quanto si può dimostrare che

$$\#\{\text{numeri primi} \leq \text{N}\} \to \frac{N}{logN} \text{ per } N \to \infty$$

quindi, se N è sufficientemente grande, in un suo interno di ampiezza $log_e N$ cade mediamente un numero primo. In media si esegue un numero di tentativi polinomiale in $n = |I|$, cioè polinomiale rispetto all'istanza di input. Vediamo il codice:

PRIMO(n)
s = sequenza casuale di n-2 bit;
P = 1s1; //lo voglio dispari e lungo n, quindi nè il primo nè bit deve essere 0
while(TestMR(P) == 0)
P = P + 10; //sommo 2 (bin) al numero composto trovato (+1 diventa pari)
return p;

8 Cifrari Storici

8.1 Cifrario Affine

Il cifrario affine è un cifrario che lavora con funzioni simili a questa sotto riportata, dove a,b sono due numeri interi arbitrari.

$$pos(y) = [a \cdot pos(x) + b]_{mod\ m}$$

Decifrare Cerchiamo di decifrare il messaggio:

$$pos(x) = \left[\frac{pos(y) - b}{a}\right]_{mod\ 26}$$

Ma non è detto che esista l'inverso di a mod m.
Vediamo in esempio:
$a = 13$
$b = 0$
$m = 26$
$pos(B) = 1 \to pos(B_y) = 13$ In questo caso quando andrò a dividere per 13 mod 26, tutte le lettere in posizione pari finiscono nello stesso posto. Questa operazione ha una sola soluzione solo se MCD(13,26=1.
Potrei quindi prendere una permutazione arbitraria
es: ABCD...z
k = TRD...A
ed ho quindi 26! chiavi, ma sono complicate da ricordare ed ho anche un cifrario estremamente facile da forzare. Basta fare un attacco provando le caratteristiche della lingua con cui è scritto: es. se la q appare tantissime volte magari la q vale la lettera a.

Cifrari Polialfabetici Si arriva quindi a pensare a dei cifrari polialfabetici (usati già da Augusto). Si poneva una sequenza lunga quanto il messaggio:
m =appu...afia
k = rqwy...pocl
Si calcolava la distanza tra le due lettere (es. $|a - r| = \alpha$) e si trasmetteva il risultato ottenuto:
crittogramma = 9 1 3 21 13 ...
Questo cifrario fu usato anche dagli inglesi, la vera forza è quello di cambiare frequentemente al frase che si usa per cifrare.

Il cifrario di Leon Battista Alberti L'idea di questo cifrario è attribuita a Leon Battista Alberti. L'idea, di realizzazione piuttosto semplice è stata poi migliorata e riutilizzata dai tedeschi nella Seconda Guerra Mondiale.

Struttura L'oggetto per cifrare è costituito da due dischi. L'alfabeto presente sul disco esterno è un alfabeto ridotto al quale sono stati aggiunti dei numeri. L'alfabeto sul disco interno è grande quanto l'alfabeto di quello esterno. Ci si mette d'accordo con il destinatario sulla posizione iniziale dei dischi, se nel messaggio metto un numero questo mi segnala un cambiamento di chiave.

$$\begin{array}{rccccccccc} m: & A & ... & ... & B & 3 & \_ & \_ & M \\ c: & C & ... & ... & S & H & \_ & \_ & T \end{array}$$

Il messaggio A.....S è stato cifrato con il messaggio C.....S mentre il messaggio $3\_\_M$ va interpretato nel seguente modo: "Il terzo (3) carattere dal numero dovrà corrispondere nella posizione della A con la lettera indicata". Nel nostro esempio dovremo ruotare i dischi in modo che la corrispondenza tra i due dischi sia $A \to M$.

Cifrario Matrice di Alfabeti Si costruisce una matrice M come la seguente:

$$\begin{array}{cccccccccc} A & B & C & D & ... & ... & S & ... & Z \\ B & C & D & E & ... & ... & T & ... & A \\ C & D & E & F & ... & ... & ... & ... & B \\ ... & & & & & & & & \\ ... & & & & & & & & \\ Z & A & B & C & ... & ... & ... & ... & Y \end{array}$$

Si decide una chiava di cifratura k:

$$\begin{array}{rccccccccc} k: & ... & ... & T & ... & ... & ... & B & ... \\ m: & ... & ... & S & ... & ... & ... & S & ... \\ C: & ... & ... & M[T][S] & ... & ... & ... & M[B][S] & ... \\ C: & ... & ... & T & ... & ... & ... & R & ... \end{array}$$

Attacchi Poichè la chiave si ripete uguale io posso dividere il messaggio in n pezzi ed eseguire attacchi classici sui cifrari monoalfabetici. Inoltre, posso eseguire una crittoanalisi statistica, cercare cioè la frequenza della comparsa delle lettere o dei gruppi di lettere e sostituirle con le lettere che compaiono con stessa probabilità nella lingua in cui è stato scritto il messaggio.

Cifrari a trasposizioni Questi metodi di cifratura sfruttano una trasposizione delle lettere già presenti nel messaggio al fine di evitare qualsiasi tecnica di crittoanalisi statistica.
La chiave di decifrazione sarà quindi la trasposizione.

$k:$	2	1	4	3	|	2	1	4	3	| ...
$m:$	A	L	F	A	|	B	E	T	O	| ...
$C:$	L	A	A	F	|	E	B	A	T	| ...

9 Cifrari Perfetti (indecidibile)

Definizione 6. *Un cifrario perfetto vuol dire che*

$$P(M = m|C = c) = P(M = m)\ \forall m, c$$

Teorema 2. *In un cifrario perfetto, il numero di chiavi N_k è $\geq$ del numero di messaggi N_m*

Dimostrazione. Supponiamo che $N_m > N_k$ e che la probabilità $P(M = m) > 0$. Esistono quindi un crittogramma $c \in C$ che non può essere immagine di nessuna funzione di cifratura $\forall k \in K$, cioè

$$P(M = m|C = c) = 0$$

, ma in questo caso il cifrario non sarebbe perfetto, poichè la $P(M = m) = 0$. □

9.1 One Time Pad

Questo cifrario, uno fra i più famosi cifrari perfetti, si basa sull'operatore binario XOR.

$$\begin{array}{ccccc} 0 & \oplus & 0 & = & 0 \\ 1 & \oplus & 1 & = & 1 \\ 0 & \oplus & 1 & = & 1 \\ 1 & \oplus & 0 & = & 1 \end{array}$$

È bene ricordare le proprietà di quest'operatore, in particolare modo l'associatività, che ci porta a poter dire che

$$x \oplus y \oplus y = x \oplus 0 = x$$

Codificare un messaggio Vediamo ora come codificare un messaggio:

$$\begin{array}{ccccccc} m & = & m_1 & m_1 & ... & m_s \\ k & = & k_1 & k_2 & ... & k_s \\ c & = & c_1 & c_2 & ... & c_s \end{array}$$

Dove il messaggio m e la chiave k sono stringhe binarie (composte cioè solo da 0 e 1) e $c_i = m_i \oplus k_i$.

Decodifica Per decodificare il messaggio ci basterà mettere in XOR ogni carattere del messaggio con il relativo carattere della chiave.

$$c_i \oplus k_i = (m_i \oplus k_i) \oplus k_i = m_i$$

Teorema 3. *Se tutti i messaggi hanno lunghezza uguale a n,*
tutte le sequenze di n bit sono messaggi possibili,
e ponendo che k sia casuale,
il One Time Pad è un cifrario perfetto.

Dimostrazione.

$$\forall m, c \;\; P(M = m|C = c) = P(M = m)$$

$$P(M = m|C = c) = \frac{P(M = m \cap C = c)}{P(C = c)}$$

Sapendo che $\#Messaggi = \#Crittogrammi = 2^n$, poichè le chiavi sono scelte in maniera random, la $P(C = c) = \frac{1}{2^n}$, in altre parole non dipende da $M = m$, quindi

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

Il cifrario è quindi perfetto. □

Problemi di questo cifrario Il problema principale di questo cifrario è lo scambio della chiave, questa infatti deve essere lunga quanto il messaggio e non può essere riutilizzabile.
Supponiamo io voglia riutilizzare la chiave, vuol dire che ho m'_i e un m''_i entrambi cifrati con la stessa chiave, cioè

$$c'_i = m'_i k_i$$

$$c''_i = m''_i k_i$$

A questo punto il crittoanalista non dovrà far altro che fare

$$c'_i \oplus c''_i = m'_i \oplus m''_i \oplus k_i \oplus k_i$$

$$c'_i \oplus c''_i = m'_i \oplus m''_i$$

Riuscendo quindi a capire molto dei messaggi originali. Come si fa quindi a stabilire tra i due partner una chiave casuale uguale? Questo è un argomento trattato più avanti.

10 Crifrari Simmetrici

I cifrari simmetrici sono tutti quei cifrari dove il mittente MITT e il destinatario DEST usano la stessa chiave sia nella funzione di cifratura C che nella funzione di decifrazione D.

10.1 DES, Data Encryption Standard

Questo cifrario si deve ritenere oramai obsoleto, è tuttavia un ottimo esempio di cifrario simmetrico da studiare.
Il cifrario lavora con blocchi di 64 bit, ognuno dei quali viene cifrato in maniera indipendente rispetto agli altri.
La chiave è una sequenza di 64 bit, 56 bit casuali e i restanti 8 bit sono bit di controllo di parità. — QUI VA l'IMMAGINE DI COME FUNZIONA —

Attacchi Questo cifrario è vulnerabile ad attacchi di tipo Chosen Plain Text.

10.2 Evoluzioni del DES

10.2.1 Scelta di più chavi

Scelta indipendente delle sottochiavi, arrivando quindi ad avere una sequenza di $k = 16 \cdot 56 = 768$ bit casuali.

10.2.2 Cifratura Multipla

$\forall m, k_3$

$$C_{DES}(C_{DES}(m, k_1), k_2) \neq C_{DES}(m, k_3)$$

Il numero di bit casuali che avremo sarà $\#\ bit\ casuali = 112$ ma esistono tuttavia attacchi che riducono lo spazio di ricerca delle chiavi da uno spazio di cardinalità $= 2^{112}$ ad uno di cardinalità $= 2^{57}$.

10.2.3 3DES (TDEA, Triple Data Encryption Algorithm)

Nel 3DES si cifra un messaggio nel seguente modo

$$c = C_{DES}(D_{DES}(C_{DES}(m, k_1), k_2), k_1)$$

e si decifra nel seguente modo

$$m = D_{DES}(C_{DES}(D_{DES}(m, k_1), k_2), k_1)$$

10.3 Altri algoritmi

Altri algoritmi che seguono la logica del DES sono l'RC5 (Ron's Code vers. 5) e l'IDEA (International Data Encryption Algorithm).

11 Crittografia asimmetrica (a chiave pubblica)

11.1 RSA

11.1.1 Matematica su cui si basa l'RSA

Scomposizione dei numeri primi C'è da dire che questo è un problema complesso.

Funzione di Eulero

11.1.2 Come funzona l'RSA

Siano $k[priv], k[pub] = k[priv]^{-1}$ due numeri interi. Il messaggio m rappresentato come numero intero può essere inviato come segue

$$c = m^{k[pub]} \text{ mod n}$$

$$m = c^{k[priv]} \text{ mod n}$$

11.2 Crifrari Ibridi

I cifrari ibridi sono i più usati. Utilizzano la crittografia asimmetrica per scambiarsi le chiavi con cui poi verrà cifrato il messaggio con un cifrario simmetrico. Questo perchè i cifrari asimemetrici sono generalmente più lenti dei cifrari a chiave simmetrica (comodi invece per scambiarsi lunghi messaggi).
Un classico esempio potrebbe essere quello dove il mittente MITT invia al destinatario DEST una coppia

$$< c_1 = C_{RSA}(k[session], k[pub]_{DEST}), c_2 = C_{AES}(m, k[session]) >$$

Ed il messaggio verrà decifrato nel seguente ordine

$$k[session] = D_{RSA}(c_1, k[priv]_{DEST})$$

$$D_{AES}(c_2, k[session])$$

11.3 Protocolli per lo scambio di chiavi

11.3.1 Protocollo di Diffie-Hellman

Siano:
ALICE e **BOB** i due interlocutori.
EVE colui che vuole intercettare il messaggio.
Si sceglie un numero primo p molto grande (1000+ bit) e un generatore g di $\mathbb{Z}_p^* = 1, 2, 3, ..., p-1$.
Alice. Sceglie a caso $0 < x < p$ e calcola quindi

$$X = g^x \text{ mod p}$$

Invia quindi X a BOB
Bob. Sceglie a caso $0 < y < p$ e calcolo quindi

$$Y = g^y \text{ mod p}$$

Invia quindi Y ad ALICE
Alice. Ricevuto Y, calcola $k[session]$ come

$$k[session] = Y^x \text{ mod p} = (g^y)^x \text{ mod p}$$

Bob. Ricevuto X, calcola $k[session]$ come

$$k[session] = X^y \text{ mod p} = (g^x)^y \text{ mod p}$$

Attacchi Questo protocollo è soggetto agli attacchi di tipo Man-in-the-middle. EVE potrebbe infatti intercettare tutti i messaggi di ALICE e sostituirsi a lei agli occhi di BOB, e fare la stessa cosa per le comunicazioni BOB $\rightarrow$ ALICE.

11.4 Crittografia Quantistica

Non capiterà mai allo scritto su!

11.5 Crittografia Post-Quantistica

Alcuni accenni alla classe di problemi BQP (Bounded Quantum Polynomial), classe di problemi decisionali con algoritmo quantistico polinomiale con $P(errore) < \frac{1}{2}$.

11.6 Crittografia su curve ellittiche

Una curva ellittica è descritta dalla seguente equazione:

$$y^2 + axy + by = x^3 + cx^2 + dx + e$$

con $a, b, c, d, e \in \mathbb{K}$ e $(x, y) \in \mathbb{K} \times \mathbb{K}$.
Nei nostri sistemi le curve saranno semplificate con equazioni del tipo

$$y^2 = x^3 + ax + b$$

ed escluderemo i casi in cui $char(\mathbb{K}) = 2, 3$ dove

Definizione 7. *$char(\mathbb{K})$ rappresenta la caratteristica di k, cioè quante volte occorre sommare l'elemento neutro per trovare l'elemento nullo. Per $\mathbb{Z}_p$*

$$char(\mathbb{Z}_p) = 1$$

Le curve ellittiche le chiameremo $E(a,b)$ dove

$$E(a,b) = \{(x,y) \in \mathbb{R}^2 | y^2 = x^3 + ax^2 + b\}$$

Chiameremo O il punto zero (o punto all'infinito). Sempre compreso nella curva, rappresenta anche l'elemento neutro. Se si lavora in $\mathbb{R}$ O è il punto all'infinito sull'asse delle ordinate.
Richiederemo nelle nostre curve che definiamo che

$$4a^3 + 27b^2 \neq 0$$

così facendo ci assicureremo che non ci siano radici multiple, cioè posso costruire in modo univoco le tangenti ad ogni punto.

Proprietà

- La curva è simmetrica rispetto all'asse x. Cioè, sia $p = (x,y) \in E(a,b) \Rightarrow p' = (x,-y) \in E(a,b)$.
- Intersecando la curva con una retta i punti di incontro sono al più 3.
- Siano $P, Q \in E(a,b)$, la somma $P + Q$ si calcola
  - Tracciando la retta passante per P, Q
  - Determinando il terzo punto d'intersezione (tra retta e curva) R
  - Ponendo $P + Q = R$
- E(a,b) ha la struttura di un gruppo abeliano additivo.

Curve ellittiche in crittografia I campi su cui si lavora generalmente parlando di curve ellittiche in crittografia sono due. Se si parla di curve prime si lavora in $\mathbb{Z}_p$ mentre se si utilizzano curve binarie si lavora in $GF(2^m)$ con $m > 0$. Trascureremo le curve binarie.
Per noi le curve ellittiche che useremo in queste pagine saranno quindi così definite

$$E_p(a,b) = \{(x,y) \in \mathbb{Z}_p^2 \mid y^2 \text{ mod p} = (x^3 + ax + b) \text{ mod p}\} \cup \{O\}$$

con p numero primo, $a, b \in \mathbb{Z}_$ e $p \neq 2, 3$ poichè altrimenti si verificherebbe che $char(\mathbb{Z}_p) = p$.

Somma sulle nostre curve ellittiche Dato $P+Q$ si può trovare $S = P+Q$ nel seguente modo.

$$x_s = (\lambda^2 - x_p - x_q) \text{ mod p}$$
$$y_s = (-y_p + \lambda(x_p - x_s)) \text{ mod p}$$

Se $P = Q$

$$\lambda = \frac{3x_p^2 + a}{2y_p} \text{ mod p}$$

Se $P \neq Q$

$$\lambda = \frac{y_q - y_p}{x_q - x_p} \text{ mod p}$$

Ordine di $E_p(a,b)$ L'ordine, poichè non sempre esistono soluzioni in $\mathbb{Z}_\text{l}$ è al più $2p+1$. Un'approssimazione che possiamo fare è dire che l'ordine è circa p in quanto ci sono circa la metà che hanno radici in $\mathbb{Z}_\text{l}$.

Teorema 4 (di Hasse)**.** *L'ordine N di una curva ellittica prima $E_p(a,b)$ verifica la seguente disequazione*

$$|N - (p+1)| \geq 2\sqrt{p}$$

Moltiplicazione Scalare Sia $P \in E_p(a,b)$, $kP = P + ... + P$ si può calcolare in $\Theta(\log k)$ raddoppi e somme di P.
Il metodo è analogo al calcolo delle potenze visto precedentemente.

11.6.1 Protocollo di Diffie-Helmann per lo scambio di chiavi su CE

11.6.2 Scambio di messaggi cifrati

Algoritmo di Koblitz

Algoritmo per lo scambio del messaggio cifrato (di El-Gamal)

Sicurezza della crittografia su CE

12 Zero Knowledge

Cos'è Serve per dimostrare che P (prover) ha qualcosa senza esibirlo a V (verifier).

Caratteristiche

- Completezza: se il segreto di P è vero, V ne accetta sempre la dimostrazione
- Correttezza: se l'affermazione di P è falsa, V può essere convinto che l'affermazione è vera solo con una bassa probabilità ($\frac{1}{2^k}$ con k grande a piacere)
- Conoscenza 0: se l'affermazione di P è vera, nessuno verifier (anche se disonesto) può acquisire alcuna informazione su questa, salvo la sua veridicità.

12.1 Protocollo di Fiat-Shamir

P sceglie

- $n = pq$ con p, q numeri primi
- $s < n, s \in \mathbb{N}$

Calcola $t = s^2$ mod n e pubblica la coppia $< n, t >$ mantenendo invece segreta la tripla $< p, q, s >$.

1. V chiede a P di iniziare una iterazione
2. P genera un intero random $r < n$;
   calcola $U = r^2$ mod n e comunica U a V
3. V genera un intero random $e \in 0, 1$. Comunica e a P.
4. P calcola $z = rs^e$ mod n $= \begin{cases} r & \text{se e} = 0 \\ z = rs \text{ mod n} & \text{se e} = 1 \end{cases}$
   Comunica z a V
5. V calcola $x = z^2$ mod n $= \begin{cases} r^2 & \text{se e} = 0 \\ r^2 s^2 & \text{se e} = 1 \end{cases}$
   Se $x = ut^e$ n itera il passo 1. altrimenti blocca e non verificare V.

13 Algoritmo di Euclide Esteso

```
Function Extended_Euclid(a,b)
    if (b = 0) then return <a, 1, 0>
    else
            <d', x', y'> = Extended_Euclid(b, a mod b);
            <d, x, y> = <d', y', x' - ⌊a/b⌋y' >
            return <d, x, y>
```

14 Definizioni Utili

14.1 Generatore

$a \in \mathbb{Z}_n^*$ è un generatore di $\mathbb{Z}_n^*$ se la funzione

$$a^k \text{ mod n}$$

con $1 \leq k \leq \Phi(n)$ genera **tutti e soli** gli elementi di $\mathbb{Z}_n^*$