Il caso

Crittografia

Luciano Margara

Unibo

2022

Bit casuale

Cosa è un bit casuale ?

In prima approssimazione, un bit casuale è una cifra binaria che può assumere valore 0 con probabilità $1/2$ e valore 1 con probabilità $1/2$.

Sequenze di bit casuali

1111111111 è casuale ?

1010011010 è casuale ?

Ma se i singoli bit sono casuali e indipendenti gli uni dagli altri $Pr(1111111111) = Pr(1010011010) = \frac{1}{2^{10}}$

Sequenze di bit casuali

$s = 11001001000011111101101010001000$
$10000101101000110000100011010011000$
10011000110011
è casuale ?
s corrisponde alle prime 25 cifre di π in binario

Definizioni di casualità

- Ce ne sono molte
- Dipendono dal contesto
- Non ne esiste una accettata da tutti

Casualità in crittografia

casuale = non (facilmente)
prevedibile

Casualità

Laplace 1819

Nel gioco di testa o croce l'apparizione di testa per cento volte di seguito è considerata un evento straordinario perché le innumerevoli combinazioni che possono avverarsi in cento lanci successivi o formano sequenze regolari, nelle quali si può osservare una regola facile da comprendere, o formano sequenze irregolari che sono incomparabilmente più numerose

Kolmogorov

Una sequenza binaria h è casuale se non ammette alcun algoritmo di generazione A la cui rappresentazione binaria sia più corta di h. In sostanza il modo più "economico" per definire una sequenza casuale è assegnare la sequenza stessa.

Kolmogorov

Le cifre di π sono casuali per Kolmogorov ?

Kolmogorov

Supponiamo di avere una procedura $Random(n)$ che produce una sequenza di n bit.
Quando n cresce le sequenze prodotte da $Random$ non saranno casuali avendo una lunghezza maggiore della rappresentazione binaria della procedura $Random$.
Quindi secondo Kolmogorov non esistono generatori di numeri casuali !!!

In pratica

Cercheremo algoritmi che producano sequenze
di bit che supereranno test di casualità.
Li chiameremo generatori di bit pseudo-casuali.

In pratica

I generatori di bit pseudo-casuali sono deterministici. Per non esserlo avrebbero bisogno a loro volta di bit casuali.
Quindi producono la stessa sequenza di bit ogni volta che li invochiamo a meno che non gli forniamo un "seme" in input.
Stesso seme, stessa sequenza !

Generatore di numeri pseudo-casuali

Un generatore di numeri pseudo-casuali è un algoritmo che parte da un piccolo valore iniziale detto seme, solitamente fornito come dato di ingresso, e genera una sequenza arbitrariamente lunga di numeri. Questa a sua volta contiene una sottosequenza detta periodo che si ripete indefinitamente. In linea di principio un generatore è tanto migliore quanto più lungo è il periodo

Generatore di numeri pseudo-casuali

Questi generatori possono però essere considerati amplificatori di casualità perché se innescati da un seme casuale di lunghezza m, fornito dall'utente, generano una sequenza "apparentemente" casuale di lunghezza $n >> m$. Una inerente limitazione è che il numero di sequenze diverse che possono essere così generate è al massimo pari al numero di semi possibili, cioè 2^m nel caso binario, enormemente minore del numero complessivo 2^n delle sequenze lunghe n.

Test statistici di casualità

- test di frequenza: verifica se i diversi elementi appaiono nella sequenza approssimativamente lo stesso numero di volte
- poker test: verifica la equidistribuzione di sottosequenze di lunghezza arbitraria ma prefissata
- test di autocorrelazione: verifica il numero di elementi ripetuti a distanza prefissata
- run test: verifica se le sottosequenze massimali contenenti elementi tutti uguali hanno una distribuzione esponenziale negativa

Generatore lineare

Un generatore pseudo-casuale molto semplice che supera con successo i quattro test citati è il generatore lineare, che produce una sequenza di interi positivi $x_1, x_2, \ldots, x_n$ a partire da un seme casuale x_0 secondo la relazione

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

dove a, b, m sono interi positivi.

Generatore lineare

Affinché il generatore abbia periodo lungo m, e quindi induca una permutazione degli interi $0, 1, \ldots, m-1$, i suoi parametri devono essere scelti in modo tale che $gcd(b, m) = 1$, $(a-1)$ sia divisibile per ogni fattore primo di m, e $(a-1)$ sia un multiplo di 4 se anche m è un multiplo di 4 (valori consigliati sono per esempio $a = 3141592653$, $b = 2718281829$, $m = 232$ e seme 0

Generatore lineare: esempio

$a = 14,\ b = 7,\ m = 13$

(1 8 2 9 3 10 4 11 5 12 6 0 7 1 8 2 9)

Generatore polinomiale

$$x_i = (a_1 x_{i-1}^t + a_2 x_{i-1}^{t-1} + \cdots + a_t x_{i-1} + a_{t+1}) \mod m$$

Generatore polinomiale binario

Si calcola il valore $r = x_i/m$: se la prima cifra decimale di r è dispari il bit generato è 1, altrimenti è 0.

Generatore polinomiale: difetti

I generatori lineari e polinomiali sono particolarmente efficienti ma non impediscono di fare previsioni sugli elementi generati, neanche quando il seme impiegato è strettamente casuale. Esistono infatti algoritmi che permettono di scoprire in tempo polinomiale i parametri del generatore partendo dall'osservazione di alcune sequenze prodotte, e questo ne svela completamente il funzionamento.

Generatori basati su funzioni one-way

Le funzioni one-way sono computazionalmente facili da calcolare e difficili da invertire: cioè si conosce un algoritmo polinomiale per il calcolo di $y = f(x)$, ma si conoscono solo algoritmi esponenziali per il calcolo di $x = f^{-1}(y)$. Notiamo che si opera su numeri, quindi il costo degli algoritmi deve essere riferito alla lunghezza della rappresentazione di x: un algoritmo che richiede un numero di operazioni proporzionale al valore di x sarà dunque esponenziale.

Generatori basati su funzioni one-way

Consideriamo la sequenza

$$S = x \; f(x) \; f(f(x)) \ldots$$

ottenuta iterando l'applicazione della f un numero arbitrario di volte. Ogni elemento della S si può calcolare efficientemente dal precedente, ma non dai successivi perchè f è one-way.

Generatori basati su funzioni one-way

Se dunque si calcola la S per un certo numero di passi senza svelare il risultato, e si comunicano poi gli elementi uno dopo l'altro in ordine inverso, ciascun elemento non è prevedibile in tempo polinomiale pur conoscendo quelli comunicati prima di esso.

Test di prossimo bit

Un generatore binario supera il test di prossimo bit se non esiste un algoritmo polinomiale in grado di predire l'$(i+1)$-esimo bit della sequenza pseudo-casuale a partire dalla conoscenza degli i bit precedenti, con probabilità significativamente maggiore di $1/2$

Test di prossimo bit

I generatori che superano il test di prossimo bit sono detti crittograficamente sicuri, e si può dimostrare che superano anche i quattro test standard discussi in precedenza. Essi vengono costruiti impiegando particolari predicati delle funzioni one-way, cioè proprietà che possono essere vere o false per ogni valore di x.

Predicati hard-core

Un predicato $b(x)$ è detto hard-core per una funzione one-way $f(x)$ se $b(x)$ è facile da calcolare conoscendo il valore di x, ma è difficile da calcolare (o anche solo da prevedere con probabilità di successo maggiore di $1/2$) conoscendo solo il valore di $f(x)$. In sostanza la proprietà hard-core, letteralmente "nucleo duro", concentra in un bit la difficoltà computazionale della f.

Predicati hard-core

$$
\begin{array}{rcl}
x & \xrightarrow{facile} & f(x) \\
x & \xrightarrow{facile} & b(x) \\
f(x) & \xrightarrow{difficile} & x \\
f(x) & \xrightarrow{difficile} & b(x)
\end{array}
$$

Predicati hard-core: esempio

Un esempio di funzione one-way è la $f(x) = x^2 \mod n$ se n non è primo. Esempio: $n = 77$ e $x = 10$, è (polinomialmente) facile calcolare il valore $y = 10^2 \mod 77 = 23$ ma è (esponenzialmente) difficile risalire al valore di $x = 10$ dato $y = 23$.
Ora il predicato: $b(x) =$ "x è dispari" è hard-core per la funzione suddetta. Infatti il valore di x permette di calcolare immediatamente $b(x) = 1$ se x è dispari, $b(x) = 0$ se x è pari, ma il problema di calcolare $b(x)$ conoscendo solo il valore $y = x^2 \mod n$ è esponenzialmente difficile.

Predicati hard-core: esempio

$$
\begin{array}{lll}
x & \xrightarrow{facile} & x^2 \mod n \\
x & \xrightarrow{facile} & x \text{ dispari ?} \\
x^2 \mod n & \xrightarrow{difficile} & x \\
x^2 \mod n & \xrightarrow{difficile} & x \text{ dispari ?}
\end{array}
$$

Generatore Blum, Blum e Shub

Sia $n = p\,q$ il prodotto di due numeri primi grandi tali che $p \mod 4 = 3$ e $q \mod 4 = 3$, $2\lfloor p/4 \rfloor + 1$ e $2\lfloor q/4 \rfloor + 1$ siano primi e sia $x_0 = y^2 \mod n$ per un qualche y. Il generatore BBS impiega x_0 come seme, calcola una successione x_i di $m \leq n$ interi e genera in corrispondenza una sequenza binaria b_i secondo la legge:

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

$$b_i = 1 \iff x_{m-i} \quad \textit{dispari}$$

Generatore Blum, Blum e Shub

Sia $p = 83,\ q = 107,\ n = 8881,\ y = 972$

Sia $x_0 = (972)^2 \mod 8881 = 3398$

Otteniamo

$x_0 = 3398$

$x_1 = 1104$

$x_2 = 2119$

$x_3 = 5256$

$x_4 = 5626$

$x_5 = 8873$

$x_6 = 64$

$x_7 = 4096$

$x_8 = 1007$

$x_9 = 1615$

$x_{10} = 6092$

Generatore Blum, Blum e Shub

$x_0 = 3398,\ b_{10} = 0$

$x_1 = 1104,\ b_9 = 0$

$x_2 = 2119,\ b_8 = 1$

$x_3 = 5256,\ b_7 = 0$

$x_4 = 5626,\ b_6 = 0$

$x_5 = 8873,\ b_5 = 1$

$x_6 = 64,\ b_4 = 0$

$x_7 = 4096,\ b_3 = 0$

$x_8 = 1007,\ b_2 = 1$

$x_9 = 1615,\ b_1 = 1$

$x_{10} = 6092,\ b_0 = 0$

Generatore Blum, Blum e Shub

Il generatore parte da x_0 cui corrisponde b_m, e procede al calcolo degli interi $x_1, \ldots, x_m$ memorizzando i corrispondenti bit $b_{m-1}, \ldots, b_0$ che comunica poi all'esterno in ordine inverso.

Generatore Blum, Blum e Shub

Dobbiamo dimostrare che il generatore BBS supera il test di prossimo bit, ma questo è in effetti un caso particolare di un risultato generale. Poniamo che $f(x)$ sia una arbitraria funzione one-way e $b(x)$ sia un suo predicato hard-core. Indichiamo con $f^{(i)}(x)$ l'applicazione iterata della funzione per i volte consecutive, e consideriamo la sequenza binaria che si ottiene partendo da un valore arbitrario di x, calcolando $b(x)$ e $f(x)$, iterando il calcolo della f un numero arbitrario di volte, e ordinando in senso inverso i valori del predicato così ottenuti

Generatore Blum, Blum e Shub

$$b(f^{(i)}(x))\ b(f^{(i-1)}(x)) \ldots b(f^{(2)}(x))\ b(f(x))\ b(x)$$

Per quanto osservato in precedenza sulle funzioni one-way, e ricordando che il predicato scelto è hard-core, segue che ciascun elemento della sequenza supera il test di prossimo bit.

Generatore Blum, Blum e Shub

Nonostante alcuni accorgimenti di calcolo per incrementarne l'efficienza, il generatore BBS è piuttosto lento poiché il calcolo di ogni bit richiede l'esecuzione di un elevamento al quadrato di un numero di grandi dimensioni nell'algebra modulare.

Generatori basati su crittografia simmetrica

A volte si preferiscono generatori teoricamente meno sicuri ma efficientissimi e comunque a tutt'oggi inviolati costruiti utilizzando le funzioni di cifratura $C(m, k)$ sviluppate per i cifrari simmetrici, utilizzando la chiave segreta k del cifrario e sostituendo il messaggio m con un opportuno valore relativo al generatore. In tal modo l'imprevedibilità dei risultati è garantita dalla struttura stessa dei cifrari, e per questi esistono realizzazioni estremamente efficienti. Poiché i cifrari generano parole (i crittogrammi) composte da molti bit, ogni parola prodotta potrà essere interpretata come numero pseudo-casuale o come sequenza di bit pseudo-casuali.

Generatori basati su crittografia simmetrica

Vediamo un generatore di questo tipo, approvato come Federal Information Processing Standard (FIPS) negli Stati Uniti: il cifrario impiegato al suo interno è in genere una versione del DES di amplissima diffusione. Detto r il numero di bit delle parole prodotte (nel DES si ha $r = 64$), s un seme casuale di r bit, m il numero di parole che si desidera produrre, e ricordando che k è la chiave del cifrario, abbiamo:

Generatori basati su crittografia simmetrica

GENERATORE(s, m)

1 $d = r$ bit presi da data e ora attuale
2 $y = C(d, k)$
3 $z = s$
4 **for** $i = 1$ **to** m
5 $\quad x_i = C(y \oplus z, k)$
6 $\quad z = C(y \oplus x_i, k)$
7 $\quad$ comunica all'esterno x_i