La casualità è un aspetto molto importante nella sicurezza dei dati perché vengono generate casualmente chiavi di lunga durata, chiavi di sessione, challange. Si consideri che il numero di chiavi che devono essere prodotte sono veramente tante, per cui è bene che questo "modo casuale" di generazione sia efficiente in modo che l'attaccante non crei problemi.

Verranno descritti concetti di **casualità** (lancio di moneta, dadi etc..) e **pseudocasualità** (generazione deterministica da un seme iniziale. Sembra casuale ma non lo è.)

GENERAZIONE CASUALE (LEGGI SOLO):

Se lancio una monetina la probabilità che esca testa o croce è del 50%. È stato effettuato un esperimento con una macchina la quale prendeva la moneta e la lanciava ripetutamente. Sono stati fatti tantissimi lanci e, con formule fisiche, si è stabilito che se viene fatto un lancio normale della monetina mettendo ad esempio testa come parte al di sopra, la probabilità di avere testa è del 51%. La probabilità che esca invece croce è del 49%.

Se ad esempio ho una moneta che da come probabilità che esca testa 70% e croce 30%. Come si ottiene un evento tale che la probabilità è 50%-50%? Si potrebbe pensare di stabilire questa cosa con 2 lanci xx e in base ai risultati determino se è testa o croce. Cioè:

- 00 ripeto i due lanci.
- 11 ripeti i due lanci.
- 01 risultato testa.
- 10 risultato croce.

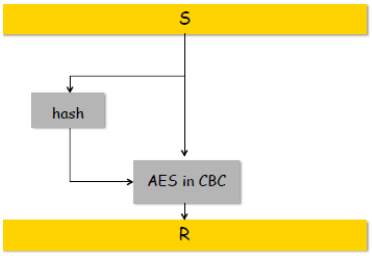
In questo modo la probabilità è 50%-50%, con 2 lanci indipendenti della stessa moneta.

Ci sono degli eventi che sfruttano eventi fisici e producono dei risultati che hanno probabilità che sono veramente casuali, ad esempio HotBits che sfrutta la radioattività.

Nei nostri computer, quali sono le sorgenti casuali che ci permettono di ottenere valori casuali? Possono essere variabili di sistema, come tempo di clock oppure numeri di serie (come indirizzo Ethernet). Numero di file su dischi o in particolare directory, spazio libero su disco, informazioni presenti nei buffer, numero di task nella coda di schedulazione del sistema operativo, le loro ID e le loro grandezze. Questi sono alcuni dei casi, ma non ne esistono tanti di sorgenti casuali. Altre sorgenti "strane" di casualità sono il contenuto di battiture su tastiera, intervalli di tempo tra la digitazione dei tasti, misure di tempi e posizione dei movimenti del cursore e/o del mouse.

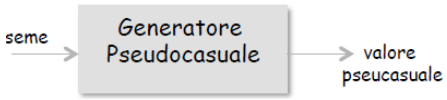
Se ho varie sorgenti, posso combinarle tra loro mediante XOR bit a bit per ottenere un valore casuale in particolare, se almeno una sorgente è casuale, allora il risultato dello XOR è casuale. Questo implica che possono esserci anche sorgenti non casuali. Questa è l'idea che viene applicata. Ad esempio:

Sia S è l'insieme di tutti i valori detti prima, per ottenere un valore R casuale, cifro questi valori in modalità CBC con AES e la chiave che do in input è l'Hash di S.



GENERAZIONE PSEUDOCASUALE:

L'idea della generazione pseudocasuale è visibile nella figura: c'è un seme (valore casuale), si utilizza un generatore pseudocasuale e l'output del generatore è la stringa casuale che ha un valore a piacere. Questa idea viene applicata per calcolare le chiavi di sessione, chiavi RSA e così via…



Il generatore più conosciuto informatica è il **generatore a congruenze lineari** che sono di questo tipo: $X_{i+1} = (a*X_i + b) \bmod m$. Si osserva che il calcolo dell' X_{i+1}-esimo valore dipende dal valore precedente. Vengono fissati a, b, m ed in questo modo genero una sequenza di valori che si basano su questa equazione. Naturalmente per poter cominciare una generazione di questo tipo ho necessità di fissare X_0 altrimenti non la generazione non potrebbe mai cominciare. Chi è X_0? È il seme che abbiamo descritto in precedenza.

Un esempio pratico (si osservi che X_0 è il seme fissato):

$$X_{i+1} = (263 \cdot X_i + 71) \bmod 100$$

$x_0 = 79$
$x_1 = (263 \cdot 79 + 71) \bmod 100 = 20848 \bmod 100 = 48$
$x_2 = (263 \cdot 48 + 71) \bmod 100 = 12695 \bmod 100 = 95$
$x_3 = (263 \cdot 95 + 71) \bmod 100 = 25056 \bmod 100 = 56$
$x_4 = (263 \cdot 56 + 71) \bmod 100 = 14799 \bmod 100 = 99$
…

Sequenza generata: 79, 48, 95, 56, 99, 8, 75, 96, 68,
36, 39, 28, 35, 76, 59, 88, 15, 16, 79, 48, 95, …

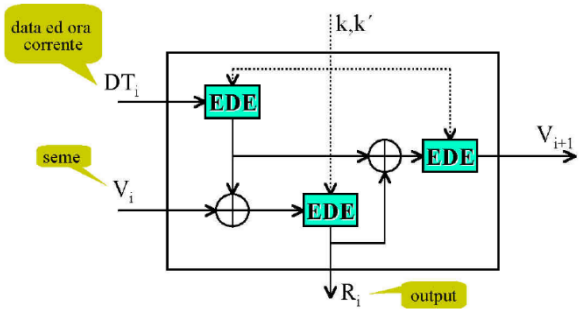
Questo generatore è implementato in tantissimi ambiti informatici, come Java, C, Microsoft Visual Basic, etc… . La popolarità di questo generatore deriva dal fatto che è molto veloce da calcolare.

Un generatore di questo tipo può essere usato per le chiavi crittografiche? No perché "in Sicurezza tutto ciò che è lineare può semplicemente rotto.", in quanto se un attaccante riuscisse a calcolare dei valori X_i avrebbe la possibilità di calcolare i valori precedenti e successivi della generazione.

Ci sono alcuni standard che permettono di usare il concetto di generazione pseudocasuale per le chiavi crittografiche con l'obiettivo di garantire sicurezza:

ANSI X9.17

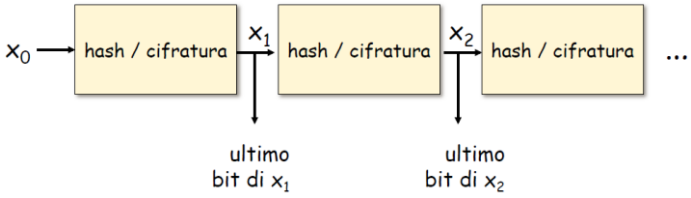
Come si osserva dalla foto, ci sono 3 EDE, che corrispondono al DES triplo in modalità **E**ncryption **D**ecryption **E**ncrypthion. Ci sono inoltre due chiavi che vengono utilizzate e sono k e k'.



Inizialmente si parte da un seme V_i che entra in input ad uno XOR bit a bit con il risultato dell'EDE che prende in input DT_i e k,k'(questo calcolo mette in pratica l'idea di casualità). Il risultato di questo XOR entra in un altro EDE insieme a k e k', e l'output R_i viene sia restituito come risultato dell'i-esima generazione, ed entra anche in input ad un altro XOR bit a bit con il risultato dell'EDE precedente calcolato su DT_i, k e k'. Il risultato dello XOR entra in input all'EDE insieme k e k' per restituire il valore V_{i+1} che è il nuovo seme che entrerà in input ad una nuova iterazione del processo descritto fino ad ora.

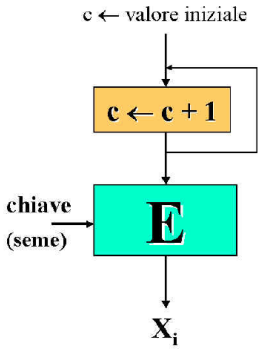
Per utilizzare la generazione pseudocasuale in crittografia, si può anche usare una catena di Hash/cifratura:

Si parte da X_0 e ne calcolo un Hash o il valore cifrato. Per il valore che ottengo, X_1 lo considero sia come ultimo bit di X_1, e ripeto il processo descritto fino ad ora. In questo caso X_0 rappresenta il seme, mentre l'output è dato dalla concatenazione degli ultimi bit di X_1, X_2 e così via.



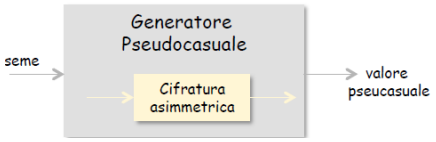
Un'altra idea potrebbe essere quella di considerare una cifratura con contatore:

Si inizia con il valore iniziale c, che viene sommato a 1 che ritorna indietro per rieseguire la somma, ma nello stesso tempo entra in input ad un algoritmo di cifratura E insieme ad una chiave che è il seme di questo algoritmo. Il risultato della cifratura corrisponde al valore X_i. La somma c= c+1 può essere sostituita dal generatore a congruenze lineari.



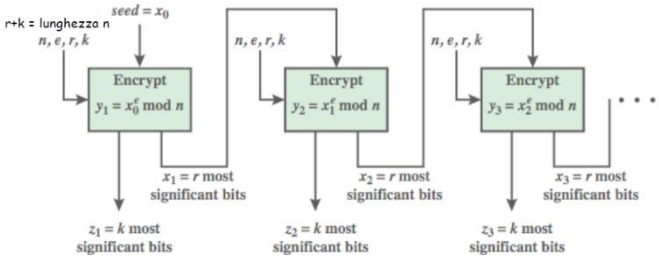
Si può cifrare l'output di un generatore (anche a congruenze lineari), anzicchè il contatore *c, c+1, c+2, ...,*

Fino ad ora abbiamo visto operazioni di cifratura simmetriche, ma la modalità di generazione pseudocasuale può essere anche applicata a cifrari asimmetrici (RSA, Diffie-Hellman ad esempio). Si deve tenere però conto che la computazione è più lenta rispetto all'utilizzo di cifrari simmetrici.



Un esempio di applicazione di generazione pseudocasuale con cifrario asimmetrico è quello di **Micali-Schnorr PRNG**:

La cifratura che viene fatta di volta in volta è un RSA. Una volta cifrato, la stringa risultante viene divisa in due parti: z_i che rappresentano i k bit più significativi che vengono dati in output, mentre x_i che rappresentano gli r bit più significativi che vengono dati in input ad un nuovo RSA. r e k vengono scelti in maniera tale che la loro somma deve essere uguale alla lunghezza di n.



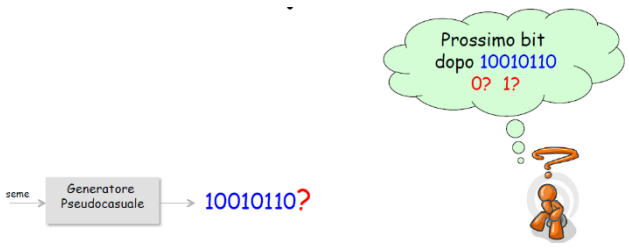
Quando diciamo che un generatore deve essere forte (valido) dal punto di vista della crittografia, s'intende che il generatore deve passare 2 test:

- Test del prossimo bit
- Test statistico (indistinguibilità) con probabilità significativamente migliore di ½

Entrambi i test sono equivalenti e verranno ora spiegati:

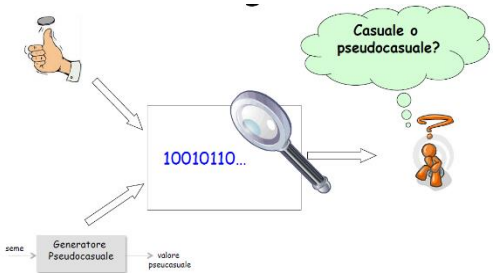
TEST DEL PROSSIMO BIT

Con un seme scelto a caso e sconosciuto all'attaccante si genera la sequenza pseudocasuale, e l'attaccante conosce la sequenza prodotta fino ad ora (nell'esempio 10010110). Se l'attaccante non riesce a determinare il valore del bit successivo, pur conoscendo tutta la sequenza prodotta fino ad ora, allora il test del prossimo bit è passato, altrimenti no. "Non riuscire a determinare il valore del bit successivo" indica che l'attaccante ha il 50% di indovinare il 50% di fallire. Se la probabilità di indovinare è del 60% allora il test fallisce.



TEST STATISTICO (INDISTINGUIBILITA')

L'attaccante conosce la sequenza prodotta fino ad un certo istante. La sequenza viene generata o in maniera casuale (ad esempio lancio di una moneta con probabilità 50%-50%) oppure mediante generazione pseudocasuale. Se l'attaccante non riesce a capire se questa sequenza viene generata casualmente o mediante generatore pseudocasuale, allora il test statistico ha successo, altrimenti no.



Si è dimostrato che il successo del test del prossimo bit si ha se e solo se si ha il successo del test statistico, e viceversa. La dimostrazione non verrà fatta.