Generazione di Numeri Casuali

Obiettivi

- Introdurre l'importanza dei numeri casuali e una loro tipica tassonomia
- Accennare a come vengono solitamente generati via hardware e/o software.

Indice

- Introduzione
- Numeri casuali veri
- Numeri pseudo-casuali
- Convalida della qualità dei numeri generati

Indice

- Introduzione
- Numeri casuali veri
- Numeri pseudo-casuali
- Convalida della qualità dei numeri generati

Utilizzo di numeri casuali

- I numeri casuali trovano largo impiego in ambiti diversi, tra i quali:
  - *simulazione* di eventi casuali
  - *gioco d'azzardo* professionale
  - *test* di sistemi digitali sia a fine produzione sia sul campo
  - *algoritmi* di tipo Monte Carlo algorithms
  - applicazioni *crittografiche*
  - …

Applicazioni crittografiche

- La sicurezza dei sistemi crittografici dipende da alcuni dati segreti, noti alle persone autorizzate ma sconosciuti e impredicibili ad altri.
- Per ottenere l'impredicibilità richiesta, si ricorre in genere alla *randomizzazione*.

Applicazioni crittografiche – Esempi di utilizzo di numeri casuali

- Generare segreti quali:
  - Chiavi di sessione
  - Numeri primi per lo scambio di chiavi e l'esponenziazione
  - Parametri nelle firme digitali
  - numeri arbitrari da utilizzare una sola volta in una comunicazione crittografica (nonces)
  - …

Random Number Generator - RNG

- Il processo di generazione di quantità casuali è di solito indicato come *Generazione di numeri casuali*
- I moduli hardware o software coinvolti sono chiamati *Random Number Generators* (RNG)

Random Number Generator

- Un *Generatore di numeri casuali* (RNG) è un'applicazione o un dispositivo in grado di produrre una sequenza di numeri appartenenti a un intervallo [min, max], tale che i valori generati appaiano impredicibili.

[https://software.intel.com/en-us/articles/intel-digital-random-number-generator-drng-software-implementation-guide]

Caratteristiche di un RNG

- Da un punto di vista tecnico, un RNG dovrebbe avere le seguenti 3 caratteristiche:
  1. Ogni nuovo valore deve essere *statisticamente indipendente* dal valore precedente.

Caratteristiche di un RNG

2. La distribuzione complessiva dei numeri nell'intervallo deve essere *uniforme*. In altre parole, tutti i numeri sono ugualmente probabili, e nessuno è più “popolare” o appare più frequentemente di altri.

Caratteristiche di un RNG

3. La sequenza deve essere impredicibile.
   Un attaccante non può indovinare alcuni o tutti i valori di una sequenza generata.
   La non prevedibilità deve essere valida sia in avanti (valori futuri) sia all'indietro (valori passati).

Importanza della qualità degli RNG

- Una scarsa qualità nel processo di RNG può introdurre gravi vulnerabilità che potrebbero compromettere un intero sistema crittografico.
- Progettare un RNG sicuro richiede quindi un livello di cura almeno pari a quello della progettazione di qualsiasi altro elemento di un sistema crittografico!

Esempi di attacchi a RNG

- Numeri casuali “deboli” possono permettere a degli attaccanti di rompere un sistema, come è stato dimostrato, ad esempio, da:
  - Rottura dell'implementazione di SSL in Netscape
  - Predizione degli ID di sessioni Java

[I. Goldberg and D. Wagner. Randomness and the Netscape browser. *Dr Dobb's*, pages 66–70, January 1996]

[Z. Gutterman and D. Malkhi. Hold your sessions: An attack on Java session-id generation. In A. J. Menezes, editor, *CT-RSA*, LNCS vol. 3376, pages 44–57. Springer, February 2005]

Aspetti critici dei RNG

- Un RNG è particolarmente “appetibile” per un aggressore, in quanto è tipicamente un singolo componente hardware o software, isolato e facile da localizzare
- Inoltre, tali attacchi richiedono un singolo accesso al sistema sotto attacco.

Tassonomia dei numeri casuali

- I numeri casuali sono di solito raggruppati in base alla *qualità della casualità* come:
  - *TRN - Numeri casuali veri*
  - *PRN - Numeri pseudo-casuali*

Indice

- Introduzione
- Numeri casuali veri
- Numeri pseudo-casuali
- Convalida della qualità dei numeri generati

Generatori di True Random Number

- I TRNG producono un flusso di numeri veramente casuali.
- In questo caso, conoscere il circuito generatore e la storia passata dei numeri generati non è di alcun aiuto nella determinazione del numero successivo.
- I TRNG sono spesso chiamati *generatori di numeri casuali non deterministici*, poiché il numero successivo da generare non può essere predetto in anticipo.

Implementazione dei TRNG

- I TRNG sono sempre basati su una *proprietà analogica*, cioè su alcune fonti fisiche imprevedibili e al di fuori di qualsiasi controllo umano.
- Il valore analogico viene poi spesso *sbiancato* e *scalato* per produrre un insieme di numeri casuali uniformemente distribuiti.

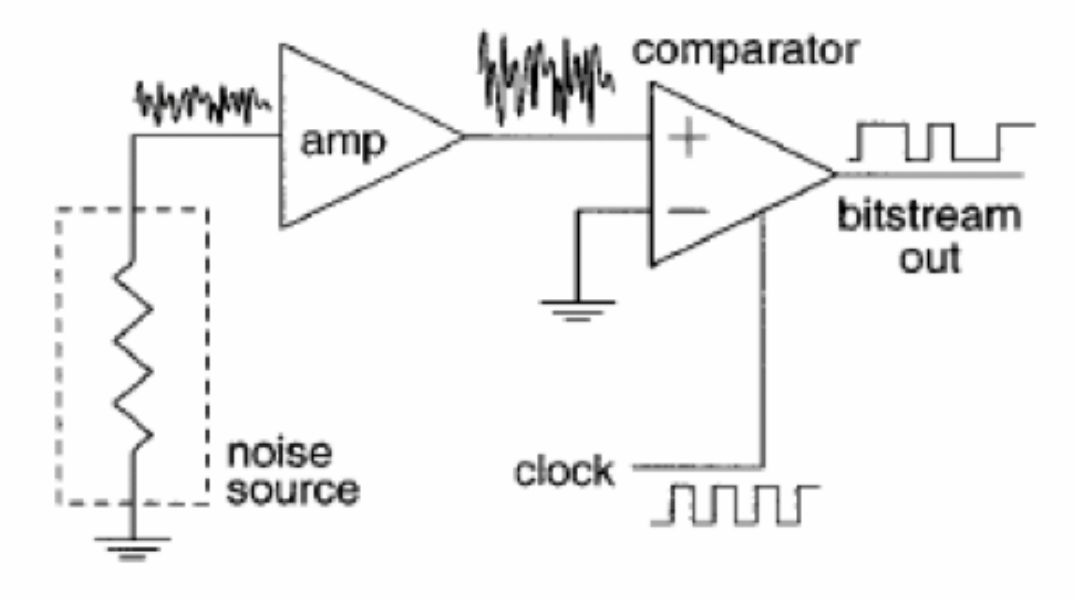
Sbiancamento

- Il processo di combinazione dell'uscita di RNG diversi, al fine di aumentare la varianza, e quindi la casualità dei numeri generati.

Sorgenti analogiche per TRNG

- Le soluzioni più adottate si basano su:
  - Amplificare il rumore generato da un resistore (rumore Johnson) o da un diodo a semiconduttore
  - Alimentare con tale rumore un comparatore o un trigger di Schmitt
  - Campionare l'uscita del comparatore per ottenere una serie di bit che possono essere utilizzati per generare numeri casuali.

Esempio



[Griffiths, Dawn, *Head First Statistics*, O'Reilly Media Inc., 2009]

Implementazioni pratiche

- In letteratura sono state proposte diverse realizzazioni pratiche TRNG, ma una loro trattazione esula dagli scopi di questa lezione
- Nell'Approfondimento 1 è riportata la soluzione adottata all'interno dei popolari processori Cortex della serie STM32 MCU F2

Implementazioni alternative

- In alcune applicazioni il TRNG viene implementato senza ricorrere a dispositivi analogici esterni personalizzati, ma semplicemente misurando alcune attività interne del sistema di elaborazione che sono misurabili e realmente casuali.

Esempi di eventi casuali

- Campionamento del valore dei secondi dal clock di Sistema al verificarsi di determinati eventi
- Campionamento di sorgenti di entropia esterne, quali i clic della tastiera, i movimenti del mouse, l'attività della rete, l’attività della telecamera e del microfono, combinate con l’ora in cui si verificano.

Esempi di eventi casuali

- La raccolta della casualità è di solito eseguita internamente dal Sistema Operativo, che fornisce API standard per accedervi (ad esempio, lettura dal file /dev/random in Linux).

G. Zvi, B. Pinkas, T. Reinman:
”Analysis of the Linux Random Number Generator” - 2006 IEEE Symposium on Security and Privacy (S&P'06) pp. 385-403

Linux Random Number Generator

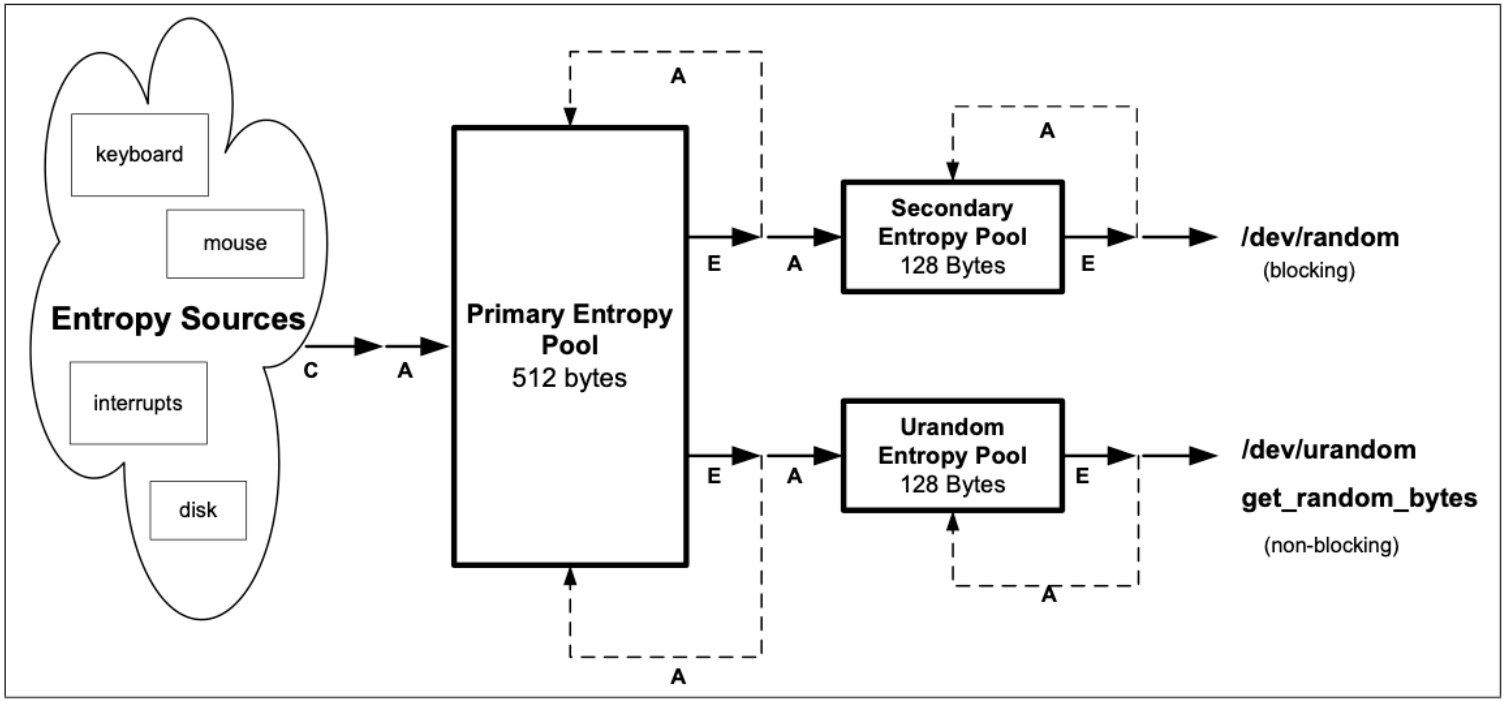


Figure 2.1: The general structure of the LRNG: Entropy is collected (C) from four sources and is added (A) to the primary pool. Entropy is extracted (E) from the secondary pool or from the urandom pool. Whenever entropy is extracted from a pool, some of it is also fed back into this pool (broken line). The secondary pool and the urandom pool draw in entropy from the primary pool.

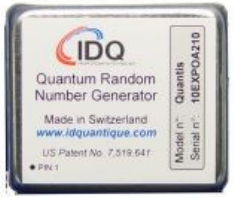
Eventi casuali - Limitazioni

- Per quanto riguarda il clock di sistema, la programmazione dei processi e altri eventi di sistema possono far sì che alcuni valori si verifichino molto più frequentemente di altri
- Quando si ha a che fare con l'entropia generata dall'uomo, i valori non si distribuiscono uniformemente nello spazio di tutti i possibili valori: alcuni valori hanno più probabilità di verificarsi rispetto ad altri, e certi valori non si verificano quasi mai nella pratica.

Quantum RNG - QRNG

- Recentemente, sono stati resi disponibili sul mercato nuovi TRNG, i cosiddetti QRNG, che estraggono i numeri casuali da *processi quantistici,* intrinsecamente non deterministici.

USB



OEM

PCIe

Quantis Appliance

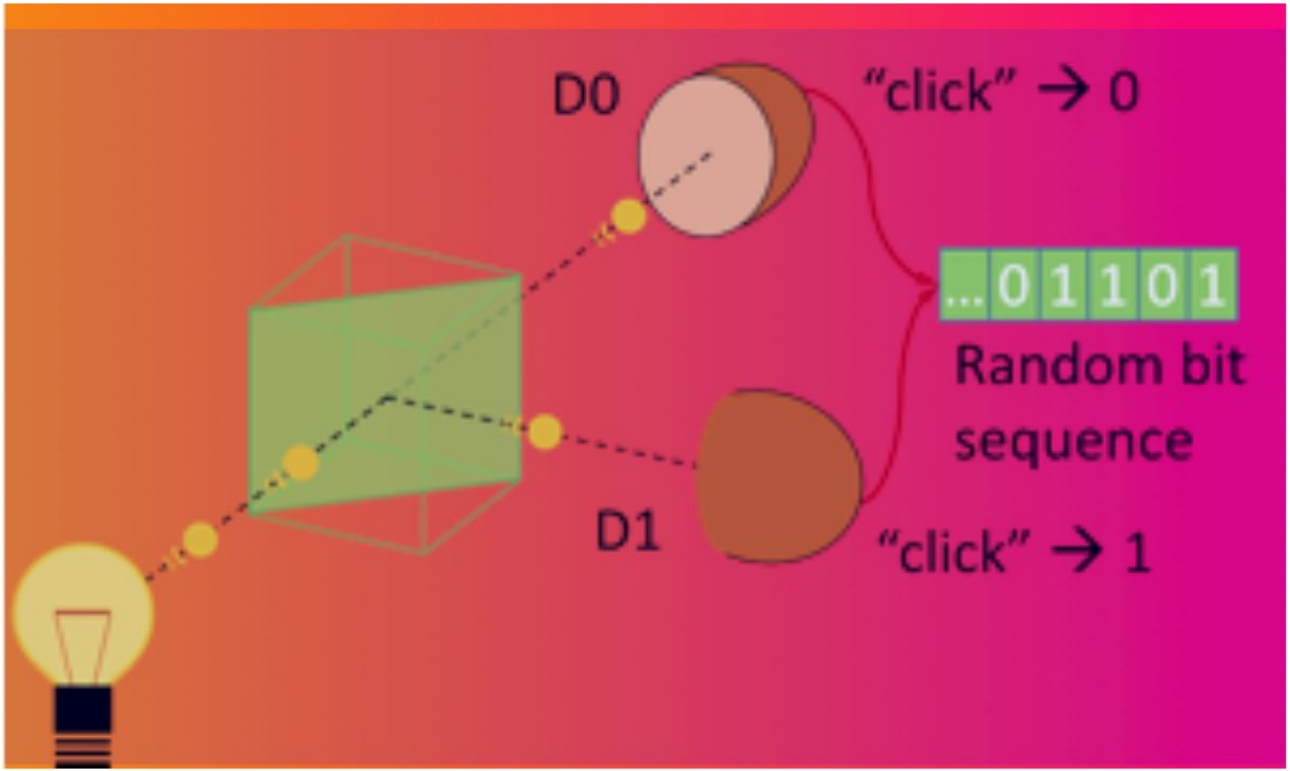
Implementazione di QRNG

- Al suo interno, un chip QRNG contiene un diodo a emissione di luce (LED) e un sensore di immagine.
- A causa del rumore quantistico, il LED emette un numero casuale di fotoni, che vengono catturati e contati dai pixel del sensore di immagine, dando una serie di numeri casuali grezzi, a cui possono accedere direttamente le applicazioni utente.

Implementazione di QRNG

- Al suo interno, un chip QRNG contiene un diodo a emissione di luce (LED) e un sensore di immagine.
- A causa del rumore quantistico, il LED emette un numero casuale di fotoni, che vengono catturati e contati dai pixel del sensore di immagine, dando una serie di numeri casuali grezzi, a cui possono accedere direttamente le applicazioni utente.

Implementazione di QRNG



Indice

- Introduzione
- Numeri casuali veri
- Numeri pseudo-casuali
- Convalida della qualità dei numeri generati

Pseudo-Random Number Generator

- Un PRNG è un algoritmo o un dispositivo hardware in grado di generare una *sequenza* di bit (o numeri) casuali, partendo da un valore iniziale, chiamato *seme* (*seed*).

Sequenze generate

- La sequenza di valori casuali generati da un PRNG:
  - si *ripete periodicamente*
  - è tipicamente *molto lunga*, ed è difficile riconoscere che la sequenza dei numeri è ordinata
  - la sua periodicità dipende dalla "*dimensione*" dello stato interno del PRNG

Lunghezza della sequenza

- I migliori algoritmi PRNG oggi disponibili hanno un periodo così ampio che questa debolezza può essere praticamente ignorata.
- Ad esempio, il Mersenne Twister MT19937 PRNG con lunghezza di parola a 32 bit ha una periodicità di $2^{19937}-1$

[Nishimura, Makoto, Matsumoto and Takuji: *Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator.* 1, January 1998, ACM Transactions on Modeling and Computer Simulation, Vol. 8]

Caveat

- Tuttavia, la perfetta conoscenza del circuito (o dell'algoritmo) di generazione e degli ultimi numeri generati permette di determinare il valore del prossimo numero che sarà generato.

Caveat

- Tuttavia, la perfetta conoscenza del circuito (o dell'algoritmo) di generazione e degli ultimi numeri generati permette di determinare il valore del prossimo numero che sarà generato.
- Questo perché un PRNG calcola il valore successivo sulla base del suo stato interno e di un algoritmo ben definito.

Consequenze

- Ne cosegue che, mentre una sequenza di valori generati mostra le proprietà statistiche di casualità (indipendenza, distribuzione uniforme), il comportamento complessivo del PRNG è del tutto prevedibile.
  - Dopo un numero sorprendentemente piccolo di osservazioni (ad esempio, 624 per Mersenne Twister MT19937), è possibile prevedere prossimo valore.

Consequenze

- Per questo motivo i PRNG sono spesso chiamati *generatori di numeri casuali deterministici*, in quanto, a partire da un particolare valore del seme, uno stesso PRNG produrrà sempre la stessa sequenza di numeri “casuali”.

Criticità del Seme

- Per garantire l'imprevedibilità, occorre prestare la massima attenzione nella generazione/scelta dei semi.
- I valori prodotti da un PRNG sono completamente prevedibili se sono noti sia il seme sia l'algoritmo di generazione.
- Poiché in molti casi l'algoritmo di generazione è disponibile al pubblico, il seme deve essere tenuto segreto e generato da un TRNG.
- Il requisito fondamentale per il buon funzionamento del PNRG è la sicurezza del seme.

Implementazioni di PRNG

- Sono possibili implementazioni diverse, basate su:
  - *Dispositivi Hardware*
  - *Programmi Software*
  - *Funzioni di Hash*

Implementazioni di PRNG

- Sono possibili implementazioni diverse, basate su:
  - *Dispositivi Hardware*
  - *Programmi Software*
  - *Funzioni di Hash*

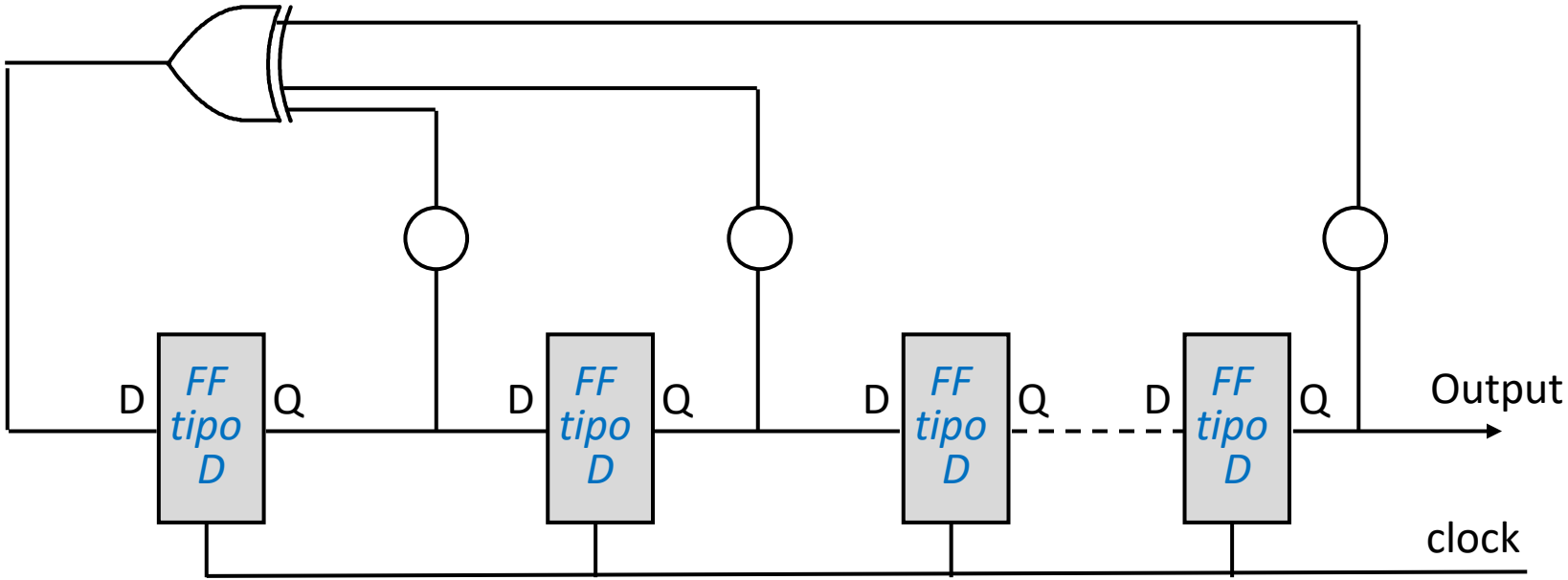
PRNG Hardware

- I PRNG hardware sono per lo più implementati ricorrendo ai cosiddetti *Autonomous Linear Feedback Shift Registers* (ALFSRs), vale a dire a degli LSFR senza ingressi e il cui polinomio caratteristico sia primitivo.

PRNG Hardware

- I PRNG hardware sono per lo più implementati ricorrendo ai cosiddetti *Autonomous Linear Feedback Shift Registers* (ALFSRs), vale a dire a degli LSFR senza ingressi e il cui polinomio caratteristico sia primitivo.

Esempio di ALFSR di n bits



Teorema

- Da un ALFSR di n bit, cambiando opportunamente il polinomio caratteristico (vale a dire i bit da inviare alla porta exor) è possibile ottenere tutti i periodi compresi tra 1 e 2^n - 1.

ALFSR a lunghezza massima

- Un ALFSR di n bit in grado di raggiungere tutti i 2^n - 1 stati possibili viene detto *a lunghezza massima* e il suo polinomio caratteristico è *primitivo.*

Caratteristiche delle Sequenze di Uscita degli ALFSR a lunghezza massima

- Partendo da uno stato diverso da 00…0, vengono raggiunti tutti i possibili stati prima di avere ripetizioni
- Se una finestra di ampiezza n è fatta scorrere sulla sequenza degli m bit in uscita, ognuna delle $2^n - 1$ possibili n-uple di bit è selezionata una e una sola volta
- Il numero degli 1 in una qualsiasi sequenza completa differisce da quello degli 0 al massimo per un'unità

Caratteristiche delle Sequenze di Uscita degli ALFSR a lunghezza massima

- In ogni sequenza completa di *m* bit:
  - vi sono tante sottosequenze di 1 quante di 0
  - metà delle sottosequenze ha lunghezza 1, un quarto ha lunghezza 2, un ottavo ha lunghezza 3, e così via
  - vi sono (*m* + 1) / 2 transizioni (da 0 a 1 o vice-versa).

Implementazioni di PRNG

- Sono possibili implementazioni diverse, basate su:
  - *Dispositivi Hardware*
  - *Programmi Software*
  - *Funzioni di Hash*

PRNG Software

- I PRNG software sono per lo più realizzati tramite programmi che implementano una operazione di ricorrenza.

Esempio di PRNG Software

- Genera il numero intero pseudocasuale successivo usando:
  - il numero intero pseudocasuale precedente
  - delle costanti intere
  - l'operazione di modulo sui numeri interi:

$$X_{n+1} = (aX_n + c) \bmod m$$

- Per iniziare, l'algoritmo richiede un seme iniziale X_0

Esempi di PRNG Software

- L'approccio precedente è uno di quelli usati più comunemente per generare numeri interi pseudocasuali

Implementazioni di PRNG

- Sono possibili implementazioni diverse, basate su:
  - *Dispositivi Hardware*
  - *Programmi Software*
  - *Funzioni di Hash*

Implementazioni di PRNG

- Sono possibili implementazioni diverse, basate su:
  - *Dispositivi Hardware*
  - *Programmi Software*
  - *Funzioni di Hash*

Pro & Contro della "Pseudo-randomicità"

- In alcuni contesti, la natura deterministica dei PRNG è un vantaggio, in quanto i PRNG permettono di generare una lunga sequenza di dati casuali in ingresso che sono ripetibili utilizzando lo stesso seme.

Pro & Contro della “Pseudo-randomicità”

- Tra gli esempi ci sono, tra gli altri:
  - In alcuni contesti di simulazione e di sperimentazione, dove si può essere interessati a confrontare i risultati forniti da approcci diversi a fronte di una stessa sequenza casuale di dati in ingresso

Pro & Contro della "Pseudo-randomicità"

- Tra gli esempi ci sono, tra gli altri:
  - In alcuni contesti di simulazione e di sperimentazione, dove si può essere interessati a confrontare i risultati forniti da approcci diversi a fronte di una stessa sequenza casuale di dati in ingresso
  - Nel collaudo di sistemi digitali, dove, al momento del test, il circuito viene eccitato con valori di ingresso casuali e i corrispondenti valori delle sue uscite vengono confrontati con quelli precedentemente calcolati tramite simulazione.

Pro & Contro della "Pseudo-randomicità"

- In altri contesti, tuttavia, il determinismo del PRNG è del tutto inaccettabile.
- Si consideri un'applicazione server che genera numeri casuali da utilizzare come chiavi crittografiche negli scambi di dati con applicazioni client su canali di comunicazione sicuri.

PRNG in crittografia

- I PRNG sono considerati insicuri da un punto crittografico.
- Tuttavia, i ricercatori hanno lavorato per risolvere questo problema creando i cosiddetti *PRNG criptograficamente sicuri* (CSPRNG).

PRNG in crittografia

- I PRNG sono considerati insicuri da un punto crittografico.
- Tuttavia, i ricercatori hanno lavorato per risolvere questo problema creando i cosiddetti *PRNG criptograficamente sicuri* (CSPRNG).

Indice

- Introduzione
- Numeri casuali veri
- Numeri pseudo-casuali
- Convalida della qualità dei numeri generati

Test per la casualità

- Vi sono diversi possibili approcci alla analisi della casualità di una sequenza di numeri

Approccio *matematico* …

- The quality of randomness can be measured in bits by the means of *Shannon entropy H(X)* of a random variable X with probability distribution p(x):

The amount of information of an elementary event $X = x$ is then $\log \frac{1}{p(x)}$. Therefore, the *average amount of information* about X is given by the expected value:

$$H(X) = \sum_x p(x) \log \frac{1}{p(x)}. \tag{5}$$

[Olivier Rioul: This is IT: A Primer on Shannon's Entropy and Information – L'Information, Seminaire Poincaré XXIII (2018) 43

Approccio *pratico* …

- Alcuni semplici modi per testare la casualità di una sequenza di bit:
  - contare il numero di "1" bit e il numero "0" bit e assicurarsi che siano approssimativamente uguali

Approccio *pratico* …

- Alcuni semplici modi per testare la casualità di una sequenza di bit:
  - contare il numero di "1" bit e il numero "0" bit e assicurarsi che siano approssimativamente uguali
  - suddividere il flusso in gruppi (ad es. di 4 bit) e assicurarsi che ogni possibile 4-tupla (0000, 0001, 0010, 0011, ... 1111) si verifichi più o meno lo stesso numero di volte
  - …

Approcci basati su *Test ad hoc*

- Sono stati sviluppati e standardizzati diversi test, di solito raggruppati in 2 classi:
  - *Test di convalida:*
    applicati a grandi sequenze di bit per misurare la qualità di un generatore di rumore casuale
  - *Test diagnostici:*
    applicati a brevi sequenze di bit (almeno 20.000 bit) per rilevare se il generatore di rumore sia stato compromesso.

Test di convalida

- Sono state create diverse suite di test per fornire requisiti e riferimenti per la loro costruzione, validazione e utilizzo dei generatori.
- Le suite più comuni sono:
  - Diehard
  - "ent"
  - NIST Statistical Test Suite (Pubblicazione speciale 800-22 Revisione 1a)

Test di convalida

- Sono state create diverse suite di test per fornire requisiti e riferimenti per la loro costruzione, validazione e utilizzo dei generatori.
- Le suite più comuni sono:
  - Diehard
  - "ent"
  - NIST Statistical Test Suite (Pubblicazione speciale 800-22 Revisione 1a)

Test diagnostici

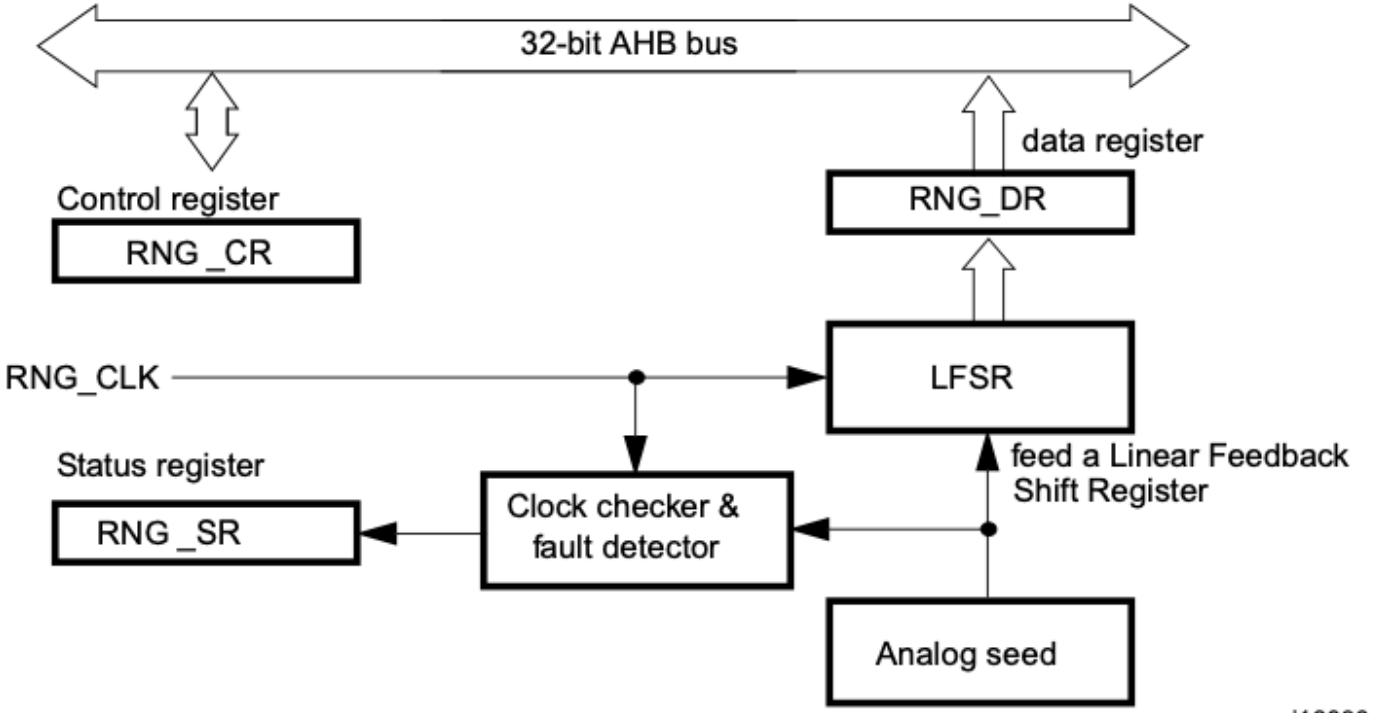
- La suite di test diagnostici più utilizzata è descritta nello standard FIPS 140-2
  - Facile da implementare nei sistemi embedded
  - Rilevamento rapido (adatto sia per la diagnostica a livello di avvio sia per quella a tempo di esecuzione)

Generazione di Numeri Casuali

Implementazione di un TRNG nella serie STM32 MCU F2

- La periferica TRNG implementata sulla serie STM32 F2 si basa su un circuito analogico che
- Questo circuito genera un rumore analogico continuo che alimenta un LFSR per produrre un numero casuale a 32 bit.
- Il circuito analogico è costituito da diversi oscillatori ad anello le cui uscite sono messe in XOR.
- L'LFSR è temporizzato da un clock dedicato a una frequenza costante, in modo che la qualità del numero casuale sia indipendente dalla frequenza di clock dell'MCU.
- Il contenuto dell'LFSR viene trasferito nel registro dati (RNG_DR) quando un numero significativo di semi è stato introdotto nell'LFSR.

Implementazione di un TRNG nella serie STM32 MCU F2



QRNG principle

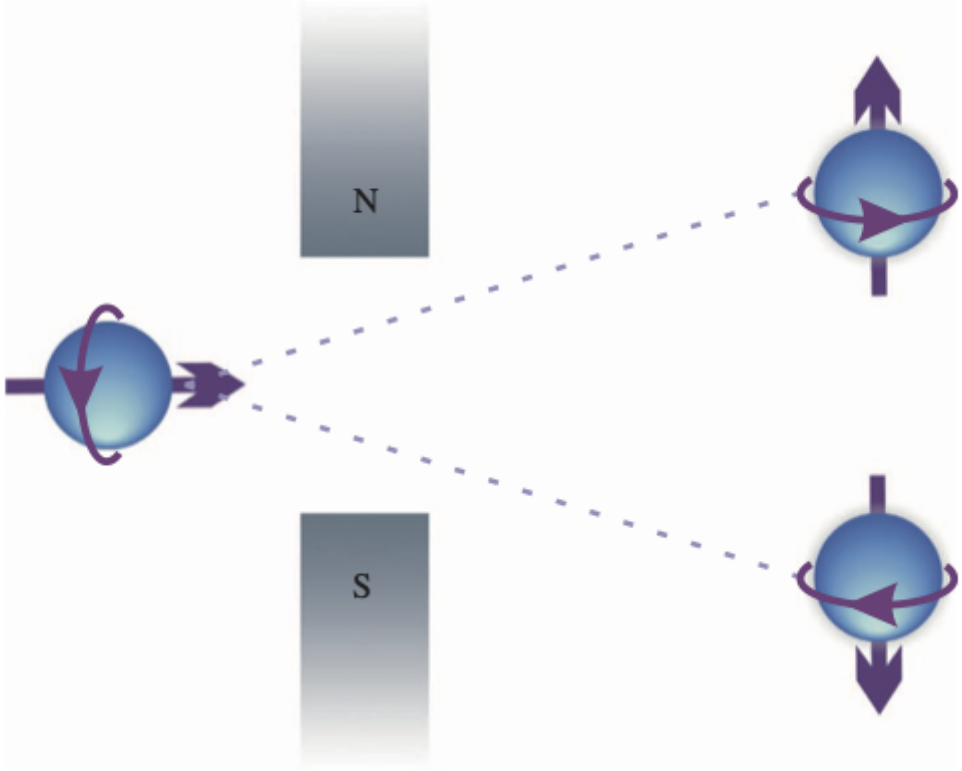
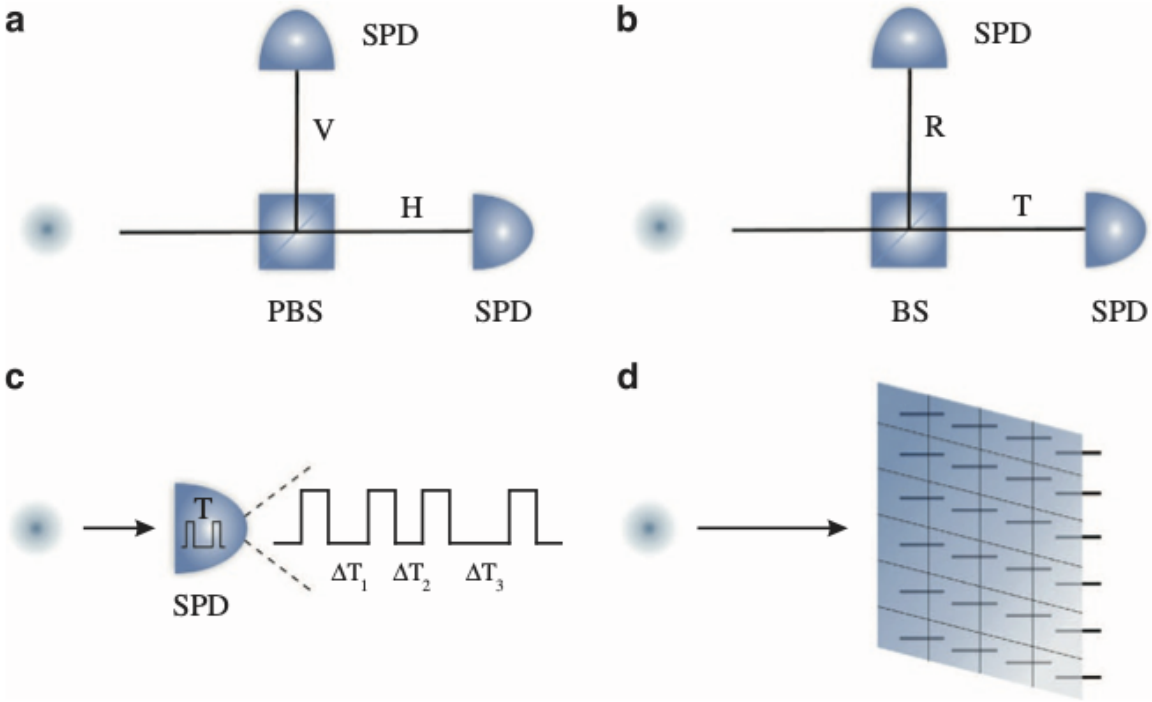


Figure 1. Electron spin detection in the Stern–Gerlach experiment. Assume that the spin takes two directions along the vertical axis, denoted by $|\uparrow\rangle$ and $|\downarrow\rangle$. If the electron is initially in a superposition of the two spin directions, $|\rightarrow\rangle = (|\uparrow\rangle + |\downarrow\rangle)/\sqrt{2}$, detecting the location of the electron would break the coherence, and the outcome ($\uparrow$ or $\downarrow$) is intrinsically random.

[Xiongfeng Ma, Xiao Yuan, Zhu Cao, Bing Qi and Zhen Zhang: “Quantum random number generation” – npj Quantum information - www.nature.com/npjqi]



[Xiongfeng Ma, Xiao Yuan, Zhu Cao, Bing Qi and Zhen Zhang: "Quantum random number generation" – npj Quantum information - www.nature.com/npjqi]

Figure 2. Practical QRNGs based on single-photon measurement. (**a**) A photon is originally prepared in a superposition of horizontal (H) and vertical (V) polarisations, described by $(|H\rangle + |V\rangle)/\sqrt{2}$. A polarising beam splitter (PBS) transmits the horizontal and reflects the vertical polarisation. For random bit generation, the photon is measured by two single-photon detectors (SPDs). (**b**) After passing through a symmetric beam splitter (BS), a photon exists in a superposition of transmitted (T) and reflected (R) paths, $(|R\rangle + |T\rangle)/\sqrt{2}$. A random bit can be generated by measuring the path information of the photon. (**c**) QRNG based on measurement of photon arrival time. Random bits can be generated, for example, by measuring the time interval, Δt, between two detection events. (**d**) QRNG based on measurements of photon spatial mode. The generated random number depends on spatial position of the detected photon, which can be read out by an SPD array.

Approfondimento

- Argomenti trattati:
  - Associazione bitvector – polinomi
  - Operazioni sui polinomi
  - Circuiti per operazioni sui polinomi
    - Moltiplicazione
    - Divisione
  - LFSR
  - ALFSR

Associazione bitvector - polinomi

Un qualsiasi vettore binario

$$R = a_m\ a_{m-1} \ldots a_2\ a_1\ a_0$$

può essere espresso tramite un *polinomio binario* di grado m

$$P(x) = a_m x^m + a_{m-1} x^{m-1} \ldots + a_1 x + a_0$$

essendo x una qualsiasi variabile di comodo.

Esempio: $10111 \rightarrow x^4 + x^2 + x + 1$

Funzione generatrice

Il *polinomio binario* P(x) viene spesso chiamato *funzione generatrice* del vettore R.

Operazioni sui polinomi

- In modo del tutto analogo a quanto avviene nell'algebra degli interi, è possibile definire sui polinomi le usuali operazioni aritmetiche.

- N.B. : Tutte le operazioni elementari sui coefficienti devono essere effettuate | 2 |

Esempio

- Dati:
  - $Q(x) = x^2 + x + 1$
  - $P(x) = x^3 + x^2 + 1$
- si ha:
  - $P(x) / Q(x) = x$
  - $P(x) \bmod Q(x) = x + 1$

Esempio

- Dati:
  - $Q(x) = x^2 + x + 1$
  - $P(x) = x^3 + x^2 + 1$
- si ha:
  - $P(x) / Q(x) = x$
  - $P(x) \bmod Q(x) = x + 1$

$$
\begin{array}{l|l}
x^3 + x^2 + 1 & x^2 + x + 1 \\
\cline{2-2}
x^3 + x^2 + x & x \\
\cline{1-1}
x + 1 &
\end{array}
$$

Polinomi Irriducibili

Definizione

Un Polinimio è detto *irriducibile* (o *primo*) se non è scomponibile in fattori

Polinomi Irriducibili

Definizione

Un Polinimio è detto *irriducibile* (o *primo*) se non è scomponibile in fattori

Esempi

$x^4 + x^3 + x^2 + x + 1$

$x^4 + x^3 + 1$

$x^4 + x + 1$

$x^2 + x + 1$

$x + 1$

Polinomi Irriducibili

Problema

$x^2 + 1$ è irreducible ?

Polinomi Irriducibili

Problema

$x^2 + 1$ è irreducible ?

Risposta

No.

x = 1 è infatti una possible soluzione, in quanto:

$$1^2 + 1 = 0$$

Ne consegue che $x^2 + 1$ può essere diviso per x + 1:

$$x^2 + 1 = (x+1)\,(x+1)$$

Teorema

- Per qualsiasi intero positivo n, esiste almeno un polinomio primitivo di grado n

Generazione di Polinomi Primitivi

Problema

- Come riconoscere o generare i polinomi primitivi?

Generazione di Polinomi Primitivi

Problema

- Come riconoscere o generare i polinomi primitivi?

Soluzione

- Non banale $\Rightarrow$ si ricorre sempre a delle tavole

Esempi di tavole

n	Esempio di Pol. Prim.	# di Pol. Prim.
1	$x + 1$	1
2	$x^2 + x + 1$	1
3	$x^3 + x + 1$	2
4	$x^4 + x + 1$	2
5	$x^5 + x^2 + 1$	6
6	$x^6 + x + 1$	6
7	$x^7 + x^3 + 1$	18
8	$x^8 + x^4 + x^3 + x^2 + 1$	16
9	$x^9 + x^4 + 1$	48
10	$x^{10} + x^3 + 1$	60
11	$x^{11} + x^2 + 1$	176
12	$x^{12} + x^6 + x^4 + x + 1$	144

n	Esempio di Pol. Prim.	# di Pol. Prim.
13	$x^{13} + x^4 + x^3 + x + 1$	630
14	$x^{14} + x^{10} + x^6 + x + 1$	756
15	$x^{15} + x + 1$	1800
16	$x^{16} + x^{12} + x^3 + x + 1$	2048
17	$x^{17} + x^3 + 1$	131072
18	$x^{18} + x^7 + 1$	262144
19	$x^{19} + x^5 + x^2 + x + 1$	524288
20	$x^{20} + x^3 + 1$	1048576
21	$x^{21} + x^2 + 1$	

Polinomi reciproci

Il polinomio *reciproco* A*(x) di un polinomio A(x) di grado m è dato da:

$$A^*(x) = x^m A(x^{-1})$$

Proprietà dei Polinomi reciproci

Se A(x) rappresenta una sequenza di n bit S_n allora A*(x) rappresenta la sequenza di bit inversa S_n*.

Esempio

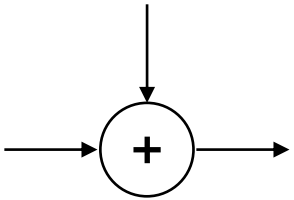
$A(x) = x^5 + x^3 + x + 1 \quad \Rightarrow \quad S_n = 101011$

$A^*(x) = x^5 + x^4 + x^2 + 1 \quad \Rightarrow \quad S_n^* = 110101$

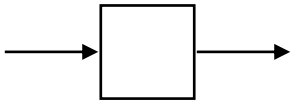
Circuiti per operazioni sui polinomi

- Nel seguito vengono presentate alcune strutture in grado di eseguire le operazioni di moltiplicazione e di divisione tra due polinomi binari, dei quali:
  - uno noto di grado r
  - uno qualsiasi, i cui coefficienti vengono trasmessi in modo seriale, a partire dal bit più significativo.

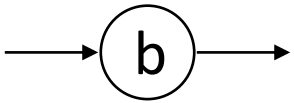
Dispositivi impiegati



sommatore modulo 2

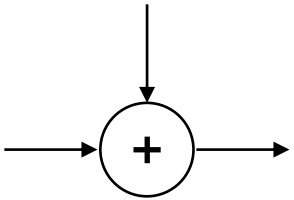


dispositivo in grado di memorizzare un bit

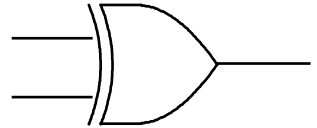


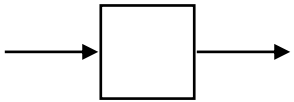
dispositivo in grado di moltiplicare per b

Dispositivi impiegati

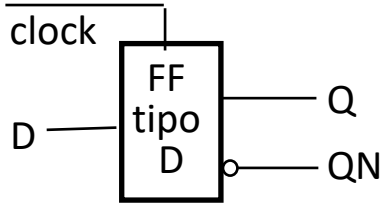


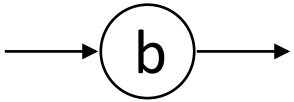
sommatore modulo 2



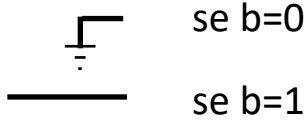


dispositivo in grado di memorizzare un bit





dispositivo in grado di moltiplicare per b



Moltiplicazione di due polinomi

Un qualsiasi polinomio

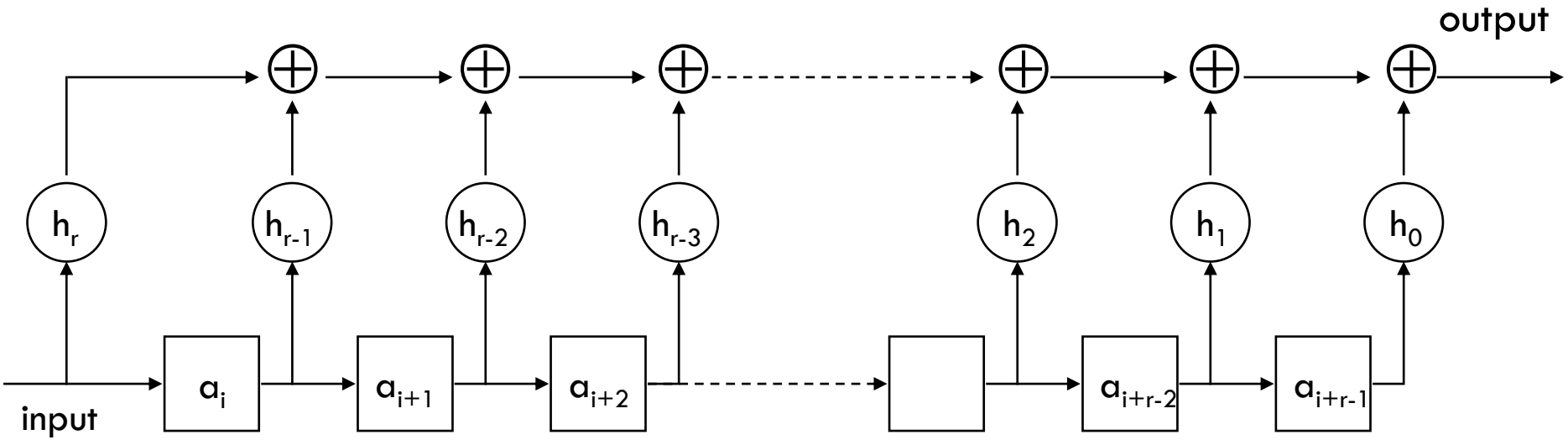
$$a(x) = a_0 + a_1 x + \ldots + a_{k-1} x^{k-1} + a_k x^k$$

può essere moltiplicato per un dato polinomio fisso

$$h(x) = h_0 + h_1 x + \ldots + h_{r-1} x^{r-1} + h_r x^r$$

tramite uno qualsiasi dei due circuiti seguenti:

Prima soluzione

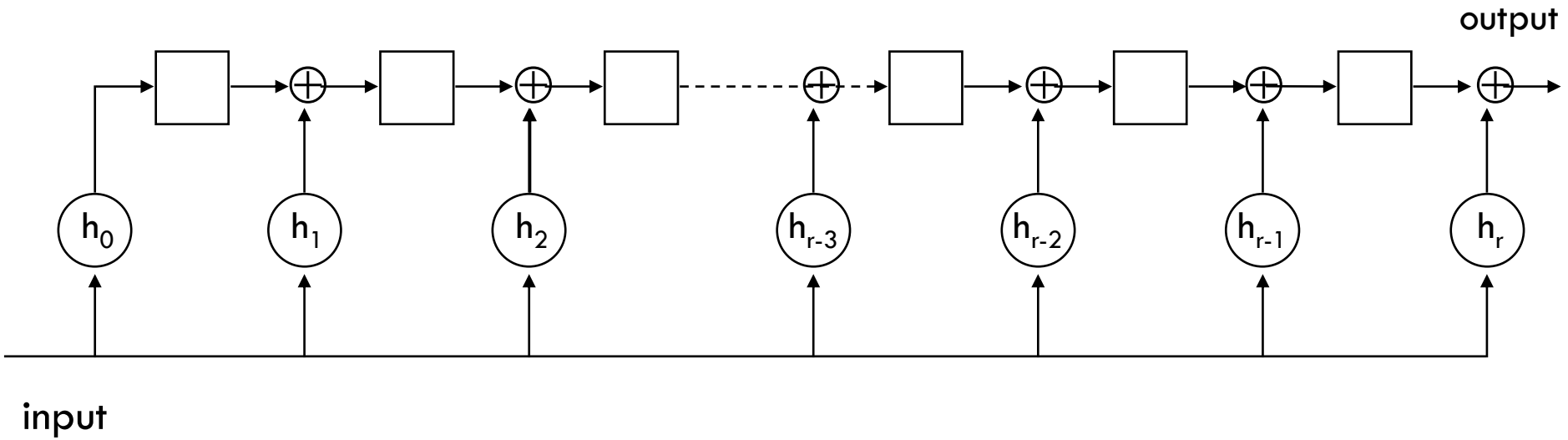


Operazione eseguita

Il prodotto vale:

$$
\begin{aligned}
a(x) \cdot h(x) \quad &= a_0 h_0 + \\
&+ (a_0 h_1 + a_1 h_0)\, x + \\
&+ (a_0 h_2 + a_1 h_1 + a_2 h_0)\, x^2 + \\
&+ \ldots + \\
&+ (a_{k-2} h_r + a_{k-1} h_{r-1} + a_k h_{r-2})\, x^{k+r-2} + \\
&+ (a_{k-1} h_r + a_k h_{r-1})\, x^{k+r-1} + \\
&+ a_k h_r\, x^{k+r}
\end{aligned}
$$

Seconda Soluzione



Divisione di due polinomi

Per dividere un polinomio qualsiasi

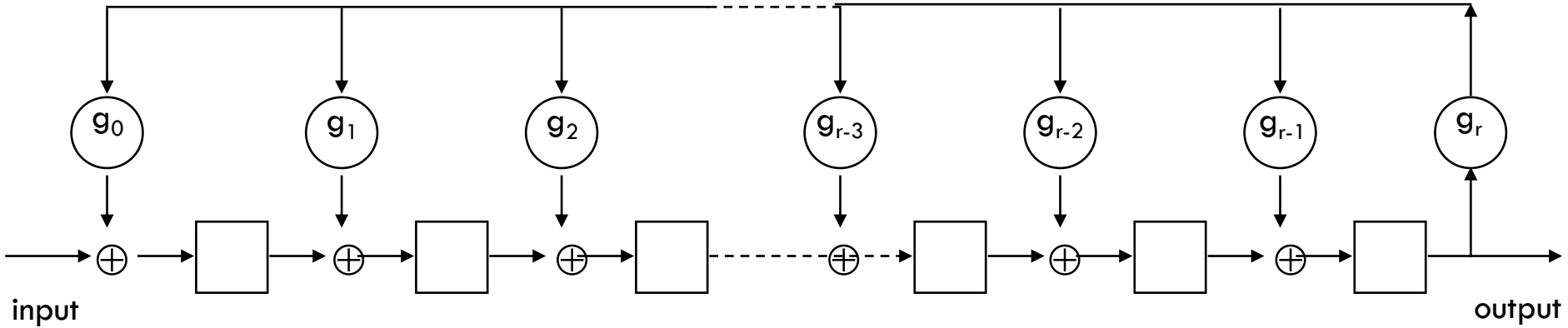
$$d(x) = d_0 + d_1 x + d_2 x^2 + \ldots + d_n x^n$$

per un polinomio fisso

$$g(x) = g_0 + g_1 x + g_2 x^2 + \ldots + g_{r-1} x^{r-1} + g_r x^r$$

si può utilizzare uno dei seguenti circuiti:

Realizzazione *modulare*

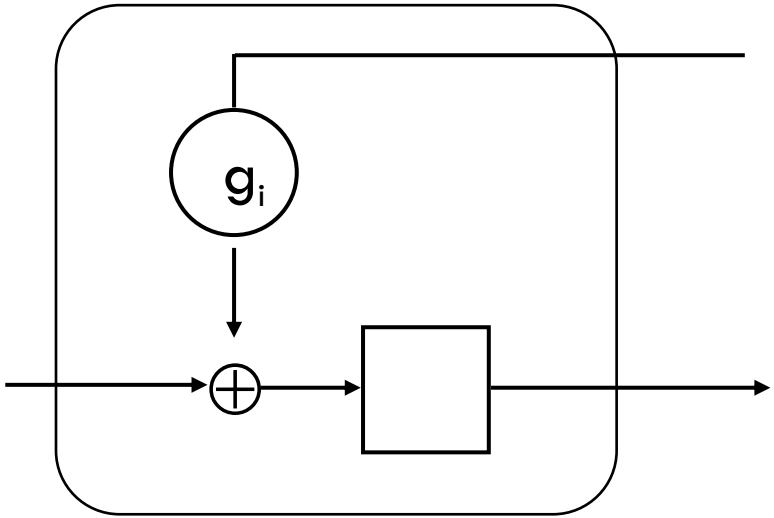


Proprietà

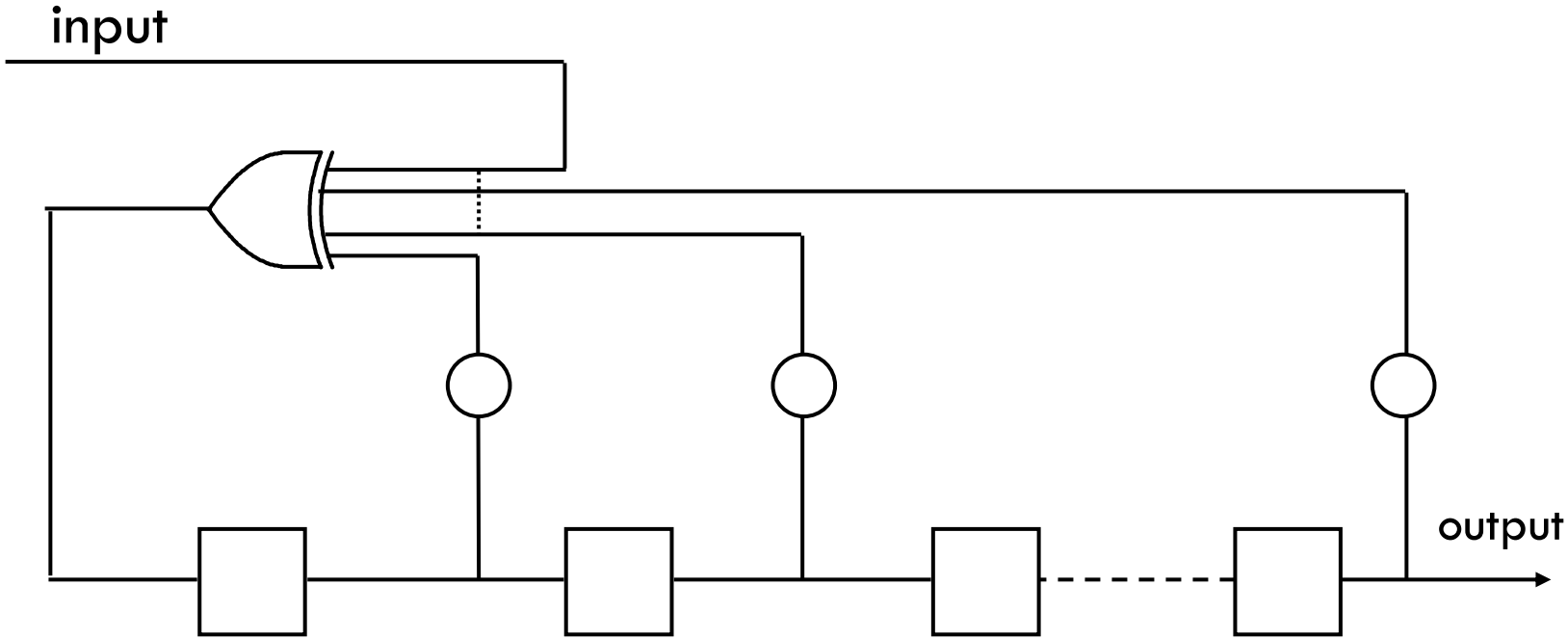
- I flip flop devono essere inizialmente azzerati
- Durante le operazioni, i bit forniti in uscita costituiscono il *quoziente* della divisione
- Al termine delle operazioni, nel registro è memorizzato il *resto* della divisione

Realizzazione modulare

- Questa prima soluzione prende il nome di *realizzazione modulare* in quanto è una cascata di m-1 celle del tipo



Realizzazione standard



Esercizio

Progettare un dispositivo
che divida per

$$g(x) = 1 + x + x^3$$

Esercizio

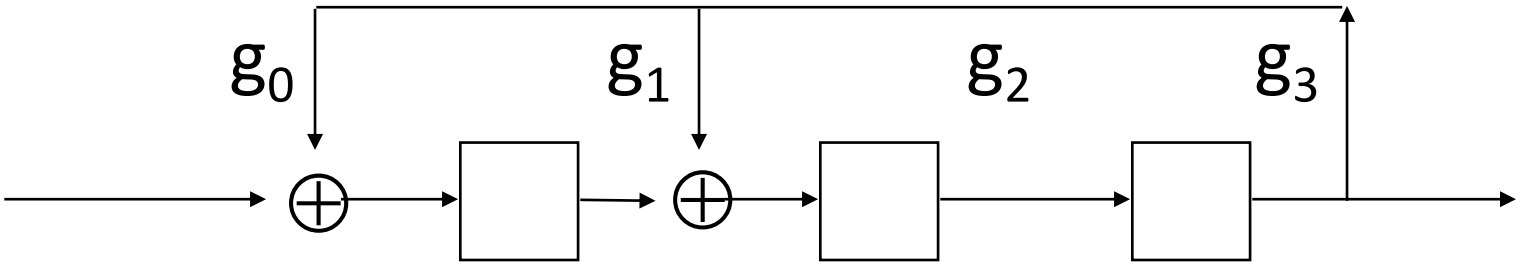
Progettare un dispositivo che divida per

$$g(x) = 1 + x + x^3$$

- Si ha:
  - $g_0 = 1$
  - $g_1 = 1$
  - $g_2 = 0$
  - $g_3 = 1$

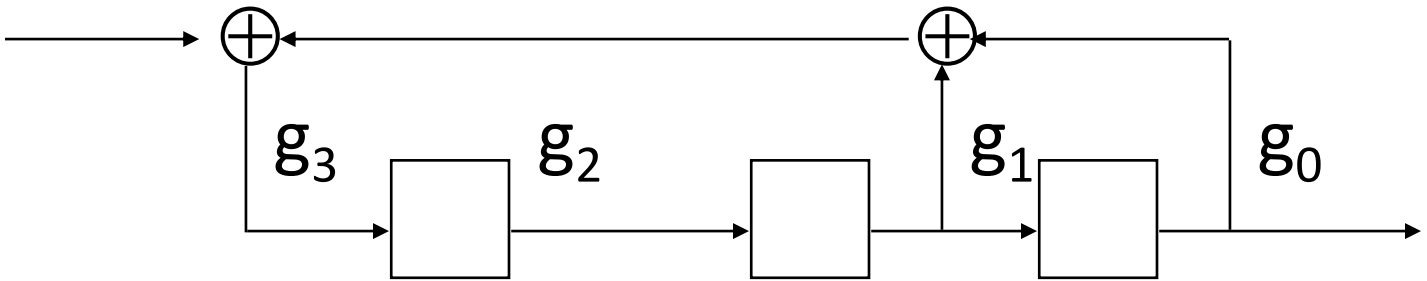
Soluzione modulare

- $g_0 = 1$
- $g_1 = 1$
- $g_2 = 0$
- $g_3 = 1$



Soluzione Standard

- $g_0 = 1$
- $g_1 = 1$
- $g_2 = 0$
- $g_3 = 1$



Circuiti *lineari*

Un qualsiasi circuito composto esclusivamente da:

- ritardatori unitari (Flip Flop di tipo D)
- sommatori modulo 2 (Exor)
- moltiplicatori modulo 2 (Shift)

è lineare, ossia soddisfa il principio di sovrapposizione:

$$f(i_1 + i_2) = f(i_1) + f(i_2)$$

LFSR

Sia la struttura modulare sia quella standard:

- essendo lineari
- essendo da uno Shift Register
- avendo una reazione

vengono solitamente chiamate *LFSR (Linear Feedback Shift Resister*).

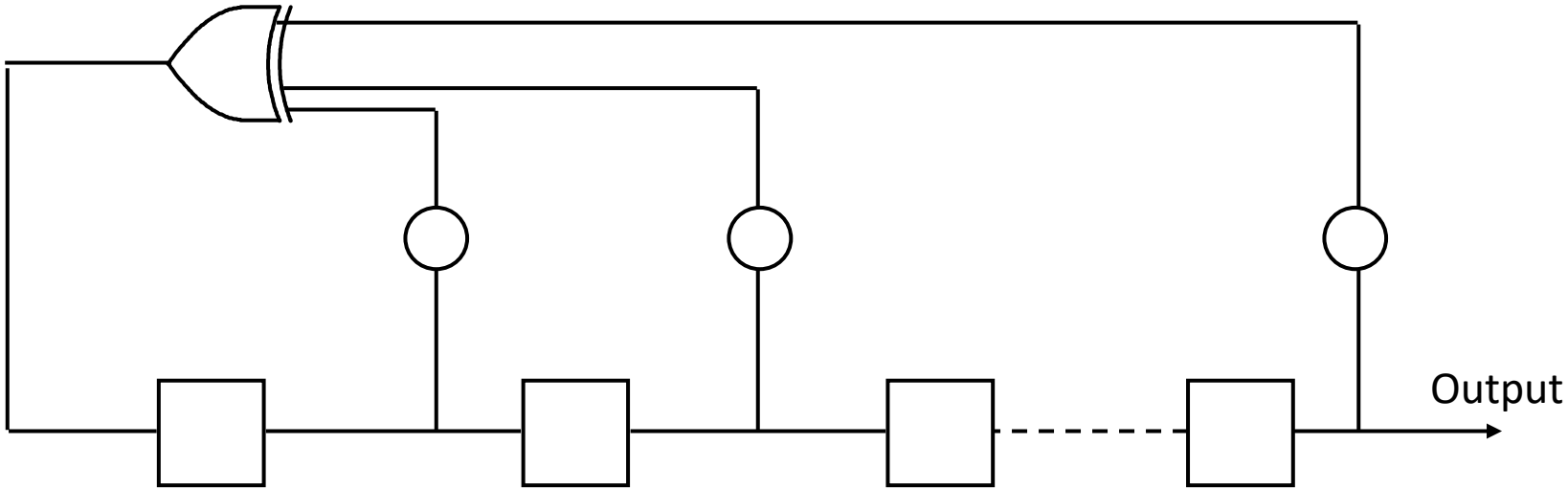
ALFSR

- Gli LFSR privi di ingressi esterni vengono chiamati *ALFSR (Autonomus LFSR).*

Implementazione di un ALFSR

- Analogamente a quanto visto per i circuiti che eseguono la divisione tra polinomi, anche gli ALFSR possono essere realizzati:
  - *in modo standard*
  - *in modo modulare*

ALFSR in forma standard



Spazio degli stati di un ALFSR

- Un ALFSR di n bit è una macchina a stati finiti caratterizzata da 2^n stati
- Lo stato 00…0 è detto *stato assorbente* e, se raggiunto, non viene più abbandonato.

Teorema

- Da un ALFSR di n bit, cambiando opportunamente il polinomio caratteristico, è possibile ottenere tutti i periodi compresi tra 1 e 2^n - 1.

ALFSR a lunghezza massima

- Un ALFSR di n bit in grado di raggiungere tutti i $2^n - 1$ stati possibili viene detto *a lunghezza massima*

Teorema

- Il polinomio caratteristico di un ALFSR a lunghezza massima è un *polinomio primitivo*

Caratteristiche delle Sequenze di Uscita degli ALFSR a lunghezza massima

- Partendo da uno stato diverso da 00…0, vengono raggiunti tutti i possibili stati prima di avere ripetizioni
- Se una finestra di ampiezza n è fatta scorrere sulla sequenza degli m bit in uscita, ognuna delle $2^n - 1$ possibili n-uple di bit è selezionata una e una sola volta
- Il numero degli 1 in una qualsiasi sequenza completa differisce da quello degli 0 al massimo per un'unità

Caratteristiche delle Sequenze di Uscita degli ALFSR a lunghezza massima

- In ogni sequenza completa di *m* bit:
  - vi sono tante sottosequenze di 1 quante di 0
  - metà delle sottosequenze ha lunghezza 1, un quarto ha lunghezza 2, un ottavo ha lunghezza 3, e così via
  - vi sono (*m* + 1) / 2 transizioni (da 0 a 1 o vice-versa).

ALFSR e Polinomi Reciproci

Due ALFSR,

- uno con polinomio caratteristico P(x)
- uno con polinomio caratteristico P*(x), reciproco di P(x)

producono, in uscita, due sequenze S_n e S_n* che sono, rispettivamente, l'una l'*inverso* dell'altra.

ASLFR come PRPG

- Le sequenze di bit generate da un ALFSR a lunghezza massima sono *pseudo-casuali* in quanto:
  - sono ripetibili deterministicamente
  - hanno proprietà analoghe a quelle delle sequenze definite *casuali*.

$\Downarrow$

Gli ALFSR sono degli ottimi generatori di sequenze pseudo-casuali

Implementazioni di CSPRNG

- Sono stati proposti diversi algoritmi CSPRNG, basati su:
  - Applicazione di un hash crittografico a una sequenza di numeri interi consecutivi
  - utilizzando un cifrario a blocchi per criptare una sequenza di numeri interi consecutivi ("counter mode")
  - XORing di un flusso di numeri generati da PRNG con testo in chiaro ("stream cipher")

Implementazioni di CSPRNG nei processori Intel®

- I processori Intel® 64 e IA-32 implementano un CSPRNG realizzato tramite un mix di componenti hardware e software.

[https://software.intel.com/en-us/articles/the-drng-library-and-manual]

[https://software.intel.com/en-us/articles/intel-digital-random-number-generator-drng-software-implementation-guide]

Intel® Digital Random Number Generator (DRNG)

- *Intel® Secure Key*, nome in codice *Bull Mountain Technology*, è il nome Intel per le istruzioni RDRAND e RDSEED delle architetture Intel® 64 e IA-32 e per l'implementazione hardware del generatore di numeri casuali digitali (DRNG) sottostante.

CSPRNG nella programmazione

- Utilizzare sempre librerie di generatori casuali crittograficamente sicuri, quali:
  - *java.security.SecureRandom* in Java 8
  - la libreria *secrets* in Python

[https://cryptobook.nakov.com/secure-random-generators]

The NIST SP800-22b statistical test set

1) **Frequency Test**: This test compares the proportion of 1's to 0's in the data. The proportion of 1's should be about half the number of bits. The test fails if there are too many or too few 1's in the bit stream.
2) **Block Frequency Test**: This test computes the proportion of 1's to 0's in a specified block size. For random data the frequency should be about half the block size. This test fails if there are too many blocks which have either too many or too few 1's.
3) **Cumulative Sums Test**: This test identifies the maximal excursion from 0 of a random walk using the values [-1, +1]. In other words, start at a point in the bit stream and move forward to the adjacent bit. If it is a "0" then SUM = SUM – 1. If it is a "1" then SUM = SUM + 1. If the bits alternated perfectly then the cumulative sum would remain low. If there are too many 1's or 0's in a row, however, the cumulative sum gets large. This test fails if the cumulative sum is either too large or too small.
4) **Runs Test**: This test counts the number of occurrences of runs of 1's. A run is defined as a continuous stream of bits of the same value bounded at the start and the end by bits of the opposite value. The expected results are more runs of shorter numbers of 1's and fewer runs of longer numbers of 1's. The test fails if there is significant deviation from the expected number of runs for any length of consecutive bits.
5) **Longest Runs Test**: This test counts the longest number of consecutive bits in each block of m bits. The test fails if there are too many consecutive 1's in the block.

The NIST SP800-22b statistical test set

6) **Rank Test**: This test divides the stream of binary bits into rows and columns of matrices. It then calculates the rank of each resulting matrix as a way of testing for linear dependence – hence too many repeated patterns. The test fails if ranks of the resulting matrices are incorrectly distributed.

7) **Discrete Fourier Transform Test**: This test examines the peak heights in the discrete Fourier transform of the sequence. The purpose is to detect repetitive patterns in the sequence. The test fails if the number of peaks exceeding a given threshold is too large.

8) **Non-overlapping Template Matching Test**: This test searches the bit stream for specific, aperiodic patterns. If the pattern is found, the search is started again just beyond the end of the pattern. If the pattern is not found, the search is started again at the next bit position. The test fails if too many occurrences of the pattern are found.

9) **Overlapping Template Test**: This test is similar to the non-overlapping template matching test except if the pattern is found, the search is continued from the next bit following the start of the pattern so that patterns which overlap are detected. Again the test fails if too many occurrences of the pattern are found.

10) **Maurer's Universal Statistical Test**: This test counts the number of bits between matching patterns in the data stream. This measure is related to how well the stream can be compressed. The test fails if the bit stream is compressible.

The NIST SP800-22b statistical test set

11) **Approximate Entropy Test**: This test compares the frequency of occurrence of all patterns of a certain bit length with the frequency of occurrence of all patterns that are one bit longer. The test fails if the difference in frequency of occurrence for the two lengths is not as expected for random data.

12) **Random Excursions Test**: This test is similar to the cumulative sum test in that a sum is calculated by taking a random walk from a point considered to be the origin and returning to that point. For each bit traversed, subtract 1 if the bit is a “0” and add 1 if the bit is a “1”. The test actually examines eight different measurements – how many times each of the sums in the set [-4, -3, -2, -1, +1, +2, +3, +4] are encountered during a random walk. The test fails if the number of times each sum is encountered does not match that predicted for random data.

13) **Random Excursions Variant Test**: This test is a more stringent variation of the random excursions test. The difference is the number of sums. This test uses a total of eighteen sums, [-9, ... -1, +1, ... +9] where the random excursions test only uses eight. The test fails when the number of times each sum occurs does not match that expected for random data.

14) **Serial Test**: This test measures the frequency of occurrence of all possible overlapping patterns of a specified bit size. In a random stream, each pattern should occur approximately the same number of times. The test fails if the number of occurrences of each pattern is not approximately the same. Note for the case of 1-bit patterns, this test degenerates to the frequency test.

15) **Lempel-Ziv Test**: This test counts the number of cumulatively distinct patterns in the sequence. It is a measure of how much the bit stream can be compressed. The test fails if the bit stream can be compressed.

16) **Linear Complexity Test**: This test calculates the size of a LFSR that would be required to produce the bit stream. The test fails if the required LFSR is too small.