

# Pseudocasualità con OpenSSL

**Alfredo De Santis**

Dipartimento di Informatica  
Università di Salerno

**ads@unisa.it**



**Maggio 2020**

# Outline

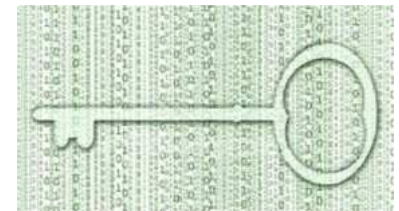
- Concetti Preliminari
- Pseudocasualità in OpenSSL

# Outline

- Concetti Preliminari
- Pseudocasualità in OpenSSL

# Utilità della "Casualità"

- Simulazione numerica
- Analisi numerica
- Decision making
- Testing computer chips
- Algoritmi probabilistici
- ...
- **Sicurezza dati**
  - Generazione chiavi, nonce, challenge
  - ...



**Casualità estremamente difficile da ottenere in pratica!**

# Alcune Sorgenti di Casualità

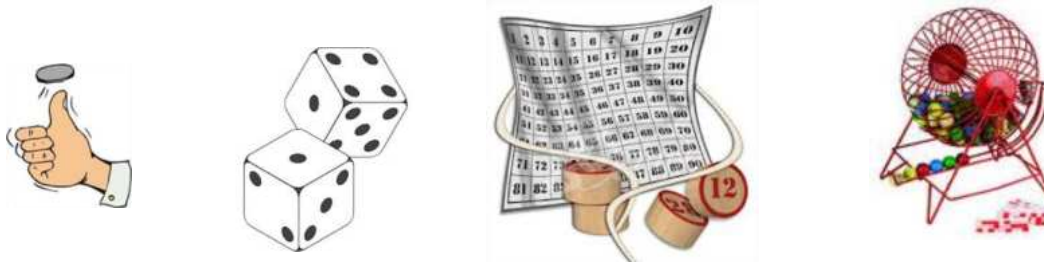
## Eventi Esterni

- Contenuto delle battiture su tastiera
- Intervalli di tempo tra la digitazione dei tasti
- Misure di tempi e posizione dei movimenti del cursore e/o del mouse  

informazione raccolta durante una sessione di lavoro  
(altrimenti si può chiedere di battere tasti a caso oppure di muovere il mouse per un po')
- Tempo di arrivo dei pacchetti su rete
- Intervalli di tempo tra interrupt

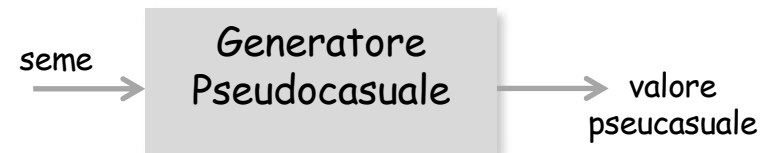
# Casualità e Pseudocasualità

## ➤ Casuale

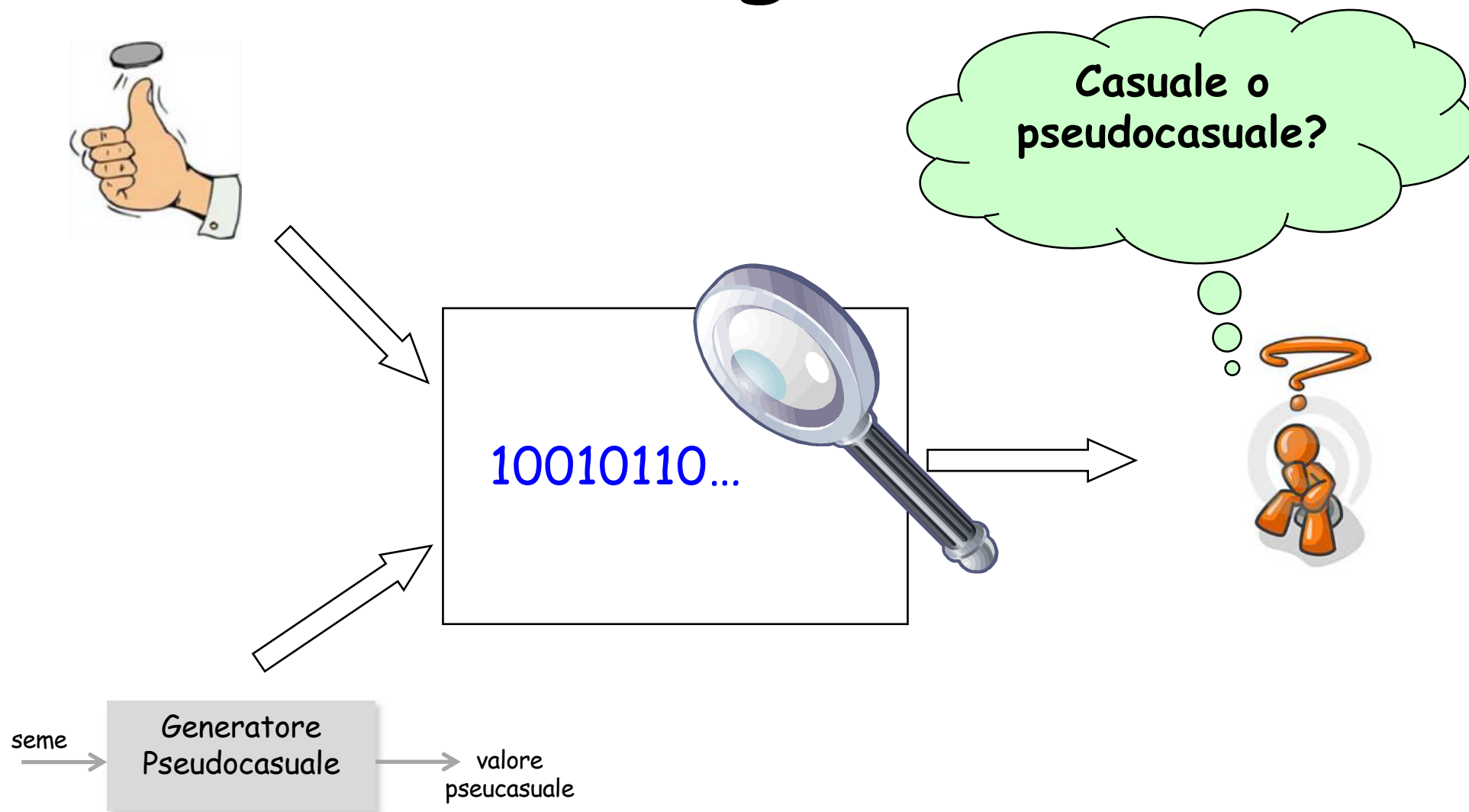


## ➤ Pseudocasuale

Generazione deterministica da un seme iniziale  
Sembra casuale ma non lo è



# Indistinguibilità



# Casualità in Sistemi ONIX

## Based Cenni

- Idea: in tutti i dispositivi sono presenti "fenomeni" o "eventi" che accadono costantemente e sono conseguenza di una certa interazione del dispositivo verso utenti o altri dispositivi
- Tali fenomeni rappresentano una sorta di *processo casuale*



# Casualità in Sistemi Operativi

## Based Cenni

- Esempio: istanti temporali associati alla digitazione dei tasti
  - Dopo aver digitato qualcosa, un utente potrebbe controllare se ci sono eventuali errori di digitazione, prendere un caffè, leggere un giornale, riflettere sulla migliore soluzione per un problema complesso, etc
- Altri esempi di processi che possono essere considerati casuali
  - Tempistica degli interrupt generati cliccando sui tasti del mouse
  - Movimenti del puntatore del mouse sullo schermo
  - Tempi associati agli eventi di I/O del disco o dei device driver
  - Informazioni inserite nei file di log (ad es., in /var/log/syslog)
  - Attività rilevata dai sensori presenti sui dispositivi
  - Tempo di arrivo dei pacchetti su rete
  - Etc

# Casualità in Sistemi UNIX

## Based Cenni

- Le sorgenti di casualità possono essere divise in due categorie
  - Appartenenti al *Kernel Space*: generano random bit a partire da eventi che avvengono nel Kernel Space
  - Appartenenti allo *User Space*: generano random bit a partire da eventi che avvengono nello User Space
- I bit casuali derivanti dal *Kernel Space* vengono memorizzati in una struttura chiamata *Entropy Pool*

# Casualità in Sistemi Unix Based Cenni

- I random bit forniti dalle sorgenti di casualità appartenenti al kernel space sono resi disponibili tramite un file speciale
  - `/dev/random`
- I random bit forniti dalle sorgenti di casualità appartenenti allo user space sono di solito resi disponibili tramite
  - *Entropy Gathering Daemon (EGD)*
  - *Pseudo Random Number Generator Daemon (PRNGD)*

# Casualità in Sistemi Unix

## Based Cenni

- Che succede se nel dispositivo di elaborazione non accadono eventi che coinvolgono l'entropy pool?
  - In `/dev/random` potrebbero non esserci sufficienti random bit
  - Una funzione che richiede l'accesso al file `/dev/random` può rimanere bloccata, finché in tale file non è presente il numero di random bit richiesto dalla funzione chiamante
- **N.B.** Di solito vengono generate solo poche centinaia di random bit al secondo
  - Se è necessario accedere ad una quantità di random bit che eccede questo tasso, non ci si può basare su `/dev/random`



# Casualità in Sistemi UNIX Based Cenni

- I sistemi UNIX-Based forniscono una sorgente non bloccante di casualità
  - Accessibile mediante il file `/dev/urandom`
  - Utilizza bit casuali forniti da `/dev/random` per inizializzare un *Cryptographically Secure Pseudo-Random Number Generator (CSPRNG)*
  - Produce flussi di bit di buona qualità crittografica



# Outline

- Concetti Preliminari
- Pseudocasualità in OpenSSL

# Pseudocasualità in OpenSSL

- OpenSSL permette di generare byte pseudocasuali
  - Possono essere scritti sullo standard output o su un file
- La generazione avviene mediante *Deterministic Random Bit Generator (DRBG)*, definiti dal *NIST SP 800-90*
  - Hash DRBG
  - HMAC DRBG
  - CTR DRBG
- OpenSSL utilizza di default un CTR DRBG, basato su AES a 256 bit
- Il DRBG usa come seme i random byte forniti da `/dev/urandom`

# Pseudocasualità in OpenSSL

- OpenSSL permette di generare byte pseudocasuali mediante il comando `rand`

## Opzioni principali del comando `rand`

```
openssl rand [options] [numbyte]
```

- **options**
  - `-rand file(s)` File usati come seme per il generatore
  - `-out file` File su cui scrivere i dati generati, che altrimenti verrebbero scritti sullo standard output
  - `-base64` I dati generati sono codificati in formato base64
  - `-hex` I dati generati sono codificati in formato esadecimale
- **numbyte** Numero di byte che si intende generare



# Pseudocasualità in OpenSSL

- OpenSSL permette di generare byte pseudocasuali mediante il comando `rand`

## Opzioni principali del comando `rand`

`openssl rand [options] [numbyte]`

- **options**

- `-rand file(s)` File usati come seme per il generatore
- `-out file` File in cui vengono salvati i dati generati, che altrimenti verrebbero scritti sullo standard output
- `-base64 I` I dati generati sono codificati in formato base64
- `-hex I` I dati generati sono codificati in formato esadecimale

- **numbyte** Numero di byte da generare

`man rand`

# Pseudocasualità in OpenSSL

## Esempio 1

- Nel seguente esempio vengono generati 12 byte pseudocasuali e scritti nel file `pseudorandom-data.bin`

```
openssl rand -out pseudorandom-data.bin 12
```

- Mediante il seguente comando è possibile visualizzare il contenuto del file `pseudorandom-data.bin`

```
$ xxd pseudorandom-data.bin  
0000000: ac36 111b 4b0a c4d2 01b2 4ae0                .6..K.....J.
```

- Mediante il seguente comando è possibile visualizzare, in formato binario, il contenuto del file `pseudorandom-data.bin`

```
$ xxd -b -g 8 -c 8 pseudorandom-data.bin | cut -d " " -f 2  
1010110000110110000100010001101101001011000010101100010011  
01001000000001101100100100101011100000
```

# Pseudocasualità in OpenSSL

## Esempio 2

- Mediante il seguente comando è possibile generare 6 random byte e codificarli in Base64
  - Questo produce una stringa di 8 caratteri

```
openssl rand -base64 6
```



```
PHRuOaeI
```

- Mediante il seguente comando è possibile generare 12 random byte e codificarli in base 64
  - Questo produrrà una stringa di 16 caratteri

```
openssl rand -base64 12
```



```
ttGNyw0twfYQZtLf
```

# Pseudocasualità in OpenSSL

## Esempio 2

### ➤ Osservazioni

- Quando i dati generati sono convertiti in Base64, la stringa prodotta avrà sempre una lunghezza multipla di quattro
- Se si desidera una stringa pseudocasuale la cui lunghezza non sia multipla di quattro, è necessario "accorciare" tale stringa mediante altri strumenti



# Pseudocasualità in OpenSSL

## Esempio 3

- Mediante il seguente comando è possibile generare una stringa pseudocasuale composta da 39 caratteri stampabili

```
openssl rand -base64 39 | cut -c1-39
```



```
MigL10P23/0OIFsiOhXH07jYYOgiG+Ud6k/kqix
```

# Pseudocasualità in OpenSSL

## Esempio 3

- Mediante il seguente comando è possibile generare una stringa pseudocasuale composta da 39 caratteri stampabili

```
openssl rand -base64 39 | cut -c1-39
```



```
MigL10P23/0OIFsiOhXH07jYYOgiG+Ud
```

Per ottenere la lista completa delle opzioni del comando cut è possibile utilizzare  
`man cut`

# Pseudocasualità in OpenSSL

## Esempio 4

- Mediante il seguente comando è possibile generare una stringa pseudocasuale composta da 39 caratteri stampabili

```
openssl rand -base64 39 | shasum | head -c39 > random.txt
```

- I dati generati da OpenSSL sono passati in pipeline alla funzione SHA-1
- L'output di tale funzione è "accorciato" per ottenere il numero di caratteri richiesto
- Il risultato è salvato nel file `random.txt`

# Pseudocasualità in OpenSSL

## Esempio 4

- Mediante il seguente comando è possibile generare una stringa pseudocasuale composta da 39 caratteri stampabili

```
openssl rand -base64 39 | shasum | head -c39 > random.txt
```

- I dati generati vengono passati alla pipeline alla funzione SHA-1
- L'output di tale pipeline viene passato alla funzione head per ottenere il numero di caratteri richiesto
- Il risultato è salvato nel file `random.txt`

Per ottenere la lista completa delle opzioni del comando `head` è possibile utilizzare `man head`



# Pseudocasualità in OpenSSL

## Esempio 5

- Nel seguente esempio, utilizzando il contenuto del file `.bash_history` come seme per il comando `rand`, vengono generati 128 byte random, codificati in Base64

```
openssl rand -rand .bash_history -base64 128
```



```
11771 semi-random bytes loaded  
BHkroMwRhGYal7Vt4x3mbCHujIkS2R4BLVoOwSgJxdaJ/3QuLaKVrNMLHoWiYNCF  
H38qyTBOrSpAagdtOvgQNdlSaZtIjlzzh57dj1Ri8ELbvTpp3395Kp3IZnhfK9t  
HnJICHSTAFL84OhIBiHW5fHyJYrL6mxxmv9NosVfz4=
```

**N.B.** Di default è utilizzato il file `/dev/urandom` come seme

# Pseudocasualità in OpenSSL

## Esempio 5

- Nel seguente esempio, utilizzando il contenuto del file `.bash_history` come seme per il comando `rand`, vengono generati 128 byte random, codificati in Base64

```
openssl rand -rand .bash_history -base64 128
```



```
11771 semi-random bytes loaded
```

```
BHkroMwRhGYal7Vt4x3mbCHujIks2R4BLVoOwSgJxdaJ/3QuLaKVrNMLHoWiYNCF  
H38qyTBOrSpAagdtOvgQNdlSaZtIj- zh57dj1Ri8ELbvTpp3395Kp3IZnhfK9t  
HnJICHSTAHL84OhIBiHW5fHyJYrL6mxx-2NosVfz4=
```

Dimensione del file usato  
come seme per il  
generatore

# Pseudocasualità in OpenSSL

## Esempio 5

- Nel seguente esempio, utilizzando il contenuto del file `.bash_history` come seme per il comando `rand`, vengono generati 128 byte random, codificati in Base64

```
openssl rand -rand .bash_history -base64 128
```



```
11771 semi-random bytes loaded
```

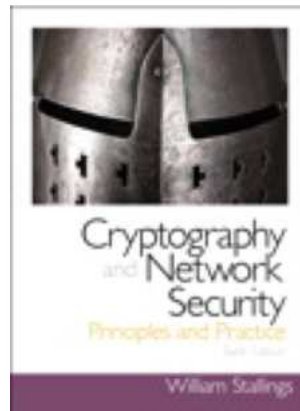
```
BHkroMwRhGYal7Vt4x3mbCHujIkS2R4BLVoOwSgJxdaJ/3QuLaKVrNMLHoWiYNCF  
H38qyTBOrRSpAagdtOvgQNdlSaZtIjlzzh57dj1Ri8ELbvTpp3395Kp3IZnhfK9t  
HnJICHSTAHL84OhIBiHW5fHyJYrL6mxxmv9NosVfz4=
```

171 caratteri + 1 di padding

N.B. Di default è utilizzato il file `/dev/urandom` come seme

# Bibliografia

- **Cryptography and Network Security** by W. Stallings, 6/e, 2014
  - Cap. 7



- **Recommendation for Random Number Generation Using Deterministic Random Bit Generators**
  - <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf>

# Bibliografia

- **Network Security with OpenSSL**  
Pravir Chandra, Matt Messier and John Viega (2002),  
O'Reilly
  - Appendix A. Command-Line Reference



- **Presentazione Lezione Corso di Sicurezza, Prof. De Santis**
  - Generatori Pseudocasuali
- **Documentazione su OpenSSL**
  - <https://www.openssl.org/docs/>