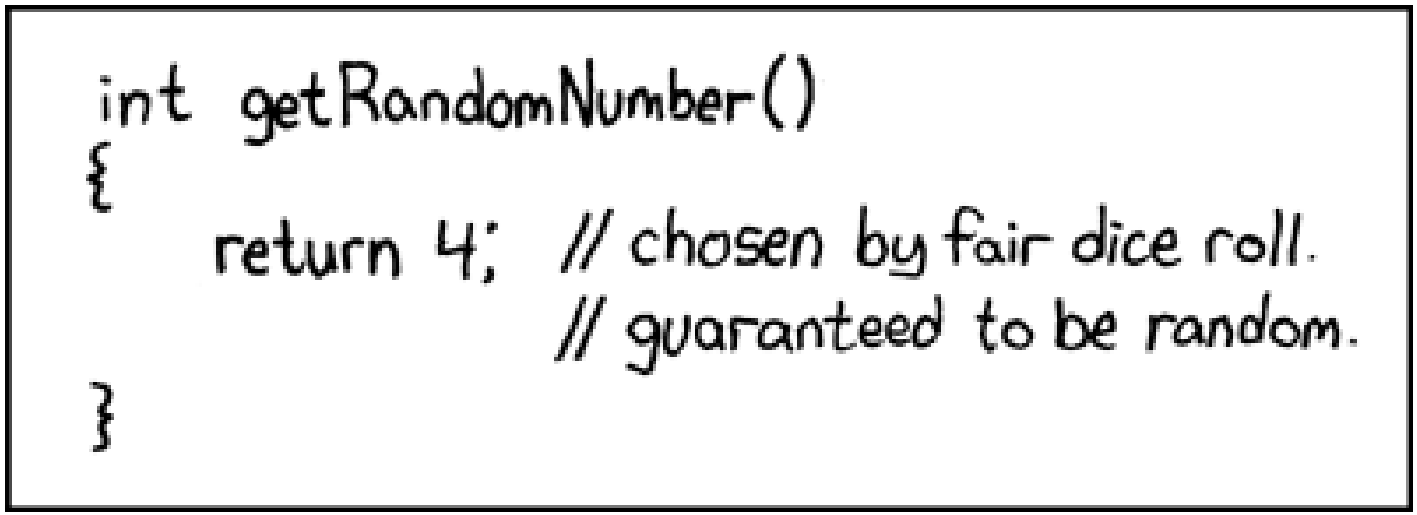
Randomness in Cryptography

JKU Linz

Erik Sonnleitner

2007

Randomness ?


int getRandomNumber()
{
return 4; // chosen by fair dice roll.
// guaranteed to be random.
}

Contents

Classifications

What classifies a computer:

- accuracy/precision
- determinism
- predictability
- traceability

What classifies randomness:

- nondeterminism
- unpredictability
- untracability

Classifications

What classifies a computer:

- **accuracy/precision**
- **determinism**
- **predictability**
- **traceability**

What classifies randomness:

- **nondeterminism**
- **unpredictability**
- **untracability**

Incompatible?

$$\text{computers} \cap \text{randomness} = \varepsilon \ ?$$

$\longrightarrow$ Find approaches to force nondeterminism within a deterministic environment.

Incompatible?

$$\text{computers} \cap \text{randomness} = \varepsilon \ ?$$

$\longrightarrow$ Find approaches to force nondeterminism within a deterministic environment.

Contents

Definition of randomness (Knu98)

- a sequence of independent random numbers
- each number was obtained completely random
- no correlations

Predictability vs entropy

Predictability:

- A fully random process is called *unpredictable*
- A fully unpredictable process *doesn't* need to be completely random.

Example: Dice roll

Entropy:

- The quantity of information within a given block of data
- The "negative logarithm of the probability of the process' most likely output" (GNP06)

Predictability vs entropy

Predictability:

- **A fully random process is called *unpredictable***
- **A fully unpredictable process *doesn't* need to be completely random.**

Example: Dice roll

Entropy:

- The quantity of information within a given block of data
- The "negative logarithm of the probability of the process' most likely output" (GNP06)

Predictability vs entropy

Predictability:

- A fully random process is called *unpredictable*
- A fully unpredictable process *doesn't* need to be completely random.

Example: Dice roll

Entropy:

- The quantity of information within a given block of data
- The "negative logarithm of the probability of the process' most likely output" (GNP06)

Entropy (2)

Gathering maximum entropy assures complete, uniformly distributed randomness and is therefore indispensable.

Example: The One-time-pad

The one-time-pad is the only known completely secure cryptographic algorithm. Security relies on:

- (the shared secret is only known by trusted partners)
- the shared secret is truly random
- every element of the random sequence is only to be used once

The securness totally relies on *useful* random numbers.
An entirely deterministic process cannot produce randomness. Hence we use mathematical approaches to gain pseudo-randomness.

Example: The One-time-pad

The one-time-pad is the only known completely secure cryptographic algorithm. Security relies on:

- **(the shared secret is only known by trusted partners)**
- the shared secret is truly random
- every element of the random sequence is only to be used once

The securness totally relies on *useful* random numbers.
An entirely deterministic process cannot produce randomness. Hence we use mathematical approaches to gain pseudo-randomness.

Example: The One-time-pad

The one-time-pad is the only known completely secure cryptographic algorithm. Security relies on:

- (the shared secret is only known by trusted partners)
- the shared secret is truly random
- every element of the random sequence is only to be used once

The securness totally relies on *useful* random numbers.
An entirely deterministic process cannot produce randomness. Hence we use mathematical approaches to gain pseudo-randomness.

Example: The One-time-pad

The one-time-pad is the only known completely secure cryptographic algorithm. Security relies on:

- **(the shared secret is only known by trusted partners)**
- **the shared secret is truly random**
- **every element of the random sequence is only to be used once**

The securness totally relies on *useful* random numbers.
An entirely deterministic process cannot produce randomness. Hence we use mathematical approaches to gain pseudo-randomness.

Example: The One-time-pad

The one-time-pad is the only known completely secure cryptographic algorithm. Security relies on:

- **(the shared secret is only known by trusted partners)**
- **the shared secret is truly random**
- **every element of the random sequence is only to be used once**

The securness totally relies on *useful* random numbers.
An entirely deterministic process cannot produce randomness. Hence we use mathematical approaches to gain pseudo-randomness.

Contents

Linear congruential generators

One of the first approaches in generating random sequences:

$$\langle X_n \rangle := x_n = (ax_{n-1} + b) \mod m$$

- x_n is taken as reminder mod m
- represents th n-th element of the pseudo-random sequence
- always liable to a maximum period $\theta < m + 1$
- maximum period will always produce the same cycle of numbers

Linear congruential generators

One of the first approaches in generating random sequences:

$$\langle X_n \rangle := x_n = (ax_{n-1} + b) \mod m$$

- x_n is taken as reminder mod m
- represents th n-th element of the pseudo-random sequence
- always liable to a maximum period $\theta < m + 1$
- maximum period will always produce the same cycle of numbers

Linear congruential generators - example

$$\langle X_n \rangle := x_n = (a x_{n-1} + b) \mod m$$

Let

- **a = b = 3,**
- **m = 11,**
- **the initial value $x_{n-1} = 0$**

$\langle X \rangle = \langle \ldots 3, 1, 6, 10, 0, \ldots \rangle$
$\rightarrow \langle X \rangle$ obtains a maximum period of 5

Congruential generators (including quadratic and plynomial ones) have been proven to be insecure due to predictability (Sch05).

Linear congruential generators - example

$$\langle X_n \rangle := x_n = (ax_{n-1} + b) \mod m$$

Let

- **a = b = 3,**
- **m = 11,**
- **the initial value $x_{n-1} = 0$**

$\langle X \rangle = \langle \ldots 3, 1, 6, 10, 0, \ldots \rangle$
$\rightarrow \langle X \rangle$ obtains a maximum period of 5

Congruential generators (including quadratic and plynomial ones) have been proven to be insecure due to predictability (Sch05).

Linear congruential generators - example

$$\langle X_n \rangle := x_n = (ax_{n-1} + b) \mod m$$

Let

- **a = b = 3,**
- **m = 11,**
- **the initial value $x_{n-1} = 0$**

$\langle X \rangle = \langle \ldots 3, 1, 6, 10, 0, \ldots \rangle$
$\rightarrow \langle X \rangle$ obtains a maximum period of 5

Congruential generators (including quadratic and plynomial ones) have been proven to be insecure due to predictability (Sch05).

Contents

Hybrid random sequence generators

...is classified by using raw data which is not completely random, but *quite* non-traceable and used in combination with ordinary randomisation algorithms.

Sustains upon:

1. direct user interaction
2. metadata about captured user interaction
3. various statistics
4. commonly used non-human-interface I/O-devices

Hybrid random sequence generators

...is classified by using raw data which is not completely random, but *quite* non-traceable and used in combination with ordinary randomisation algorithms.

Sustains upon:

1. **direct user interaction**
2. **metadata about captured user interaction**
3. **various statistics**
4. **commonly used non-human-interface I/O-devices**

User interaction

Direct interaction:

- Keystrokes
- Mouse-movements and positioning (x,y)
- etc.

Metadata about interaction:

- Inter- as well as intra-keystroke-timings
- Mouse-movement-timings and acceleration
- etc.

User interaction

Direct interaction:

- Keystrokes
- Mouse-movements and positioning (x,y)
- etc.

Metadata about interaction:

- Inter- as well as intra-keystroke-timings
- Mouse-movement-timings and acceleration
- etc.

Statistics and non-HI devices

Statistics:

- Network-, and packet statistics
- process- and timetables
- etc.

Non-human-interface I/O-devices:

- cameras
- microphones
- etc.

Statistics and non-HI devices

Statistics:

- Network-, and packet statistics
- process- and timetables
- etc.

Non-human-interface I/O-devices:

- cameras
- microphones
- etc.

Sidestep: Algebraical random generator under Linux

- Linux also uses a hybrid random number generator, which supplies two random number sources:
  - `/dev/random`: Provides a very high level of entropy and therefore gives good but limited randomness
  - `/dev/urandom`: Provides unlimited random numbers, but reuses entropy if not enough available
- The algebraical part of this generator sustains upon multiple iterations of the SHA-1 hashing-algorithm.
- Especially via `/dev/urandom` gathered entropy is re-used via re-hashing the collected data.

Sidestep: UI-driven randomness under Linux

The UI-driven part of the Linux RNG mainly uses the following input sources:

1. **Keystrokes (and timings),**
2. **Mouse movements (and timings),**
3. **Disk I/O operations,**
4. **System interrupts**

Note: Embedded Linux distributions like OpenWRT for routers often *cannot* provide good entropy, due to inexistence of input devices and harddisks; especially after resets!

Contents

True randomness

Atomic decay represents a good source for true randomness by measuring the quantity of emissions within a fixed time-frame by a Geiger-counter.

(SR07) describes the construction of a true quantum random number generator, measuring the quantum physical processes of photonic emissions in semiconductors. Provides up to 12Mbit$\cdot s^{-1}$

True randomness

Atomic decay represents a good source for true randomness by measuring the quantity of emissions within a fixed time-frame by a Geiger-counter.

(SR07) describes the construction of a true quantum random number generator, measuring the quantum physical processes of photonic emissions in semiconductors. Provides up to 12Mbit$\cdot s^{-1}$

Contents

Randomness through ordinary disk drives

Tests have shown, that

- **rotantional speed,**
- **disk spacing and**
- **(cooling) air-flow**

directly affect the disk accessing behaviour.

The algorithm measures the access-time for reading single sectors on the disk and uses the variation of these times as input source. Results still deserve the removal of correlations!

Why results are supposed to be random: Main reason for disk-jitter is found in natural air turbulences inside the drive. (DIF94, ES00)

Randomness through ordinary disk drives

Tests have shown, that

- **rotantional speed,**
- **disk spacing and**
- **(cooling) air-flow**

directly affect the disk accessing behaviour.

The algorithm measures the access-time for reading single sectors on the disk and uses the variation of these times as input source. Results still deserve the removal of correlations!

Why results are supposed to be random: Main reason for disk-jitter is found in natural air turbulences inside the drive. (DIF94, ES00)

Contents

Neumann-filtering

When a (pseudo) bitstream is gathered, the focus should be set on eliminitating correlations. One possible algorithm is the *Neumann-Filter* (Ert03).

$$f = \begin{cases} 00 \longrightarrow \epsilon \\ 11 \longrightarrow \epsilon \\ 01 \longrightarrow 0 \\ 10 \longrightarrow 1 \end{cases}$$

Neumann-filtering

When a (pseudo) bitstream is gathered, the focus should be set on eliminitating correlations. One possible algorithm is the *Neumann-Filter* (Ert03).

$$f = \begin{cases} 00 \longrightarrow \epsilon \\ 11 \longrightarrow \epsilon \\ 01 \longrightarrow 0 \\ 10 \longrightarrow 1 \end{cases}$$

Let p be the possibility of any bit x inside the stream $\langle X \rangle$ to be 1, then the possibility of getting the value 1 after using the Neumann-Filter is defined through

$$p_n = \frac{p \cdot (1-p)}{2p \cdot (1-p)} = 0.5$$

Why randomness is important

- Cryptography often *relies* on true randomness
- Randomness commonly builds the basement of key-generation processes
- Possibilities of reproduction/traceability *must not* exist

Numerous vulnerabilities and exploits have been reported, which invalidate (strong) cryptographic routines due to corrupt random sources.

Contents

How Netscape gathered entropy

```
1  global variable seed;
2
3  RNG_CreateContext()
4       (seconds, microseconds) = time of day;
5       pid = process ID;  ppid = parent process ID;
6       a = mklcpr(microseconds);
7       b = mklcpr(pid + seconds + (ppid << 12));
8        seed = MD5(a, b);
```

Early SSL implementations

The original Netscape code only took three parameters to gather entropy for key generation and exchange:

- **system ID of current process**
- **system ID of parent process**
- **current time of day**

Several papers have shown that this combination of pseudo-random sources is breakable within under one minute (Gut98).

Early SSL implementations

The original Netscape code only took three parameters to gather entropy for key generation and exchange:

- **system ID of current process**
- **system ID of parent process**
- **current time of day**

Several papers have shown that this combination of pseudo-random sources is breakable within under one minute (Gut98).

Contents

TCP sequence number prediction (Bel98)

- TCP uses stateful connections, and hence sequence numbering to assure correctness
- Both, server and client, chose their own random values as initial sequence number θ_C, θ_S

TCP 3-way-handshake:

1. C $\rightarrow$ S: SYN (θ_C)
2. S $\rightarrow$ C: SYN (θ_S), ACK (θ_C)
3. C $\rightarrow$ S: ACK (θ_S)

TCP sequence number prediction (Bel98)

- TCP uses stateful connections, and hence sequence numbering to assure correctness
- Both, server and client, chose their own random values as initial sequence number θ_C, θ_S

TCP 3-way-handshake:

1. C $\rightarrow$ S: SYN (θ_C)
2. S $\rightarrow$ C: SYN (θ_S), ACK (θ_C)
3. C $\rightarrow$ S: ACK (θ_S)

Prediction on Berkeley systems

- On Berkeley systems, sequence numbers are permanently incremented once a second by a constant amount
- Morover, they're increased by half of this amount everytime a connection is initiated
- This levels the ground for serious TCP session hijacking, interception and manipulation attacks, due to sequence number predictability.

Any questions?

Erik Sonnleitner
delta-xi.net

Thank you.