Random Bit Generators

Gianluca Dini
Dept. of Ingegneria dell'Informazione – University of Pisa
Email: gianluca.dini@unipi.it

Version: 2024-02-28

1

References (→)

- **SP 800-90A** Recommendation for Random Number Generation Using Deterministic Random Bit Generators (June 2015)
  - Specifies mechanisms for generating random bits via deterministic methods.
- **SP 800-90B** Recommendation for the Entropy Sources Used for Random Bit Generation, (January 2018)
  - Covers design principles and requirements for entropy sources (ES), devices that generate unpredictability, and NRBGs.

2

References

- **SP 800-90C** Recommendation for Random Bit Generator (RBG) Constructions (April 2016)
  - Discusses how to combine the entropy sources presented in 90B with the DRNGs presented in 90A to provide large quantities of unpredictable bits for use in cryptographic applications.
- **SP 800-22** A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications (April 2010)
  - Discusses the selection and validation of NRBGs and DRBGs.

Random Bit Generator

- DEFINITION. A Random Bit Generator (RBG) outputs a sequence of statistically independent and unbiased bits
  - Statistically independent means that the probability of emitting a bit value (1 or 0) does not depend on the previous bits
  - Unbiased means that the probability of emitting a bit value (1 or 0) is equal to 0.5

Random Bit Generators

- Random Number Generators (RNGs) can be used to generate uniformly distributed random numbers
- A random number in the interval [0, n] can be obtained by generating a bit sequence of length $\lfloor \lg n \rfloor + 1$ and converting it to an integer;
  - If the resulting number exceeds n, one possible option is to discard it and generate another random bit sequence

Mar-25 Random Generators 5

5

Random Bit Generators

- Classes of RBGs
  - True random bit generators (TRBG)
  - Pseudorandom Bit Generator (PRBG)
  - Cryprographically Secure Pseudorandom Bit Generator (CSPRBG)

Mar-25 Random Generators 6

6

True Random Bit Generators

- Based on a physical process
  - Coin flipping, rolling a dice, semiconductor noise, clock jitter, radioactive decay
- The output «cannot» be reproduced
  - Pr[flipping a coin 100 times and generate a given 100-long sequence] = $1/2^{100}$
- Classification
  - Hardware-based generators
  - Software-based generators

Mar-25 Random Generators 7

7

TRBG – Hardware-based

- Physical phenomena
  - elapsed time between emission of particles during radioactive decay
  - thermal noise from a semiconductor diode or resistor
  - the frequency instability of a free running oscillator
  - the amount a metal-insulator semiconductor capacity is charged during a fixed period of time
  - air turbulence within a sealed disk drive which causes random fluctuations in disk drive sector read latency times
  - sound from a microphone or video from a camera

Mar-25 Random Generators 8

8

TRBG – Hardware-based

Intel embeds this into their HW.

- Example: Intel Digital Random Number Generator
- Introduced in Intel CPUs since 2012
- Based on NIST SP 800-90
- Exploits thermal noise fluctuations with the CPU
- DRNG and RDRAND/RDSEED assembly instructions
- Partially documented

Mar-25 Random Generators 9

9

TRBG – Hardware-based

→ Someone can impose a behavior

- Subject to external influence and malfunction
  - Subject to observation and manipulation
- Periodic tests
- Defective generators
  - Biased: Probability of emitting a 1 is not equal to 0.5
  - Correlated: Probability of emitting a 1 depends on previous bit emitted
- De-skewing techniques: generate truly random bit sequences from the output bits of a defective generator
  - A practical technique is to pass the sequence through a cryptographically secure hash function

Manipulate this sequence so it comes back to be random again (hash function)

Mar-25 Random Generators 10

10

TRBG – Hardware-based

- Deskewing: an example
  - Suppose that i) a generator produces biased but uncorrelated bits; ii) the probability of a 1 is p, and the probability of a 0 is 1−p, where p, 0 < p < 1, is unknown but fixed. Remove biases in output bits.

- Solution
  - Group the output sequence into pairs of bits, with
    - a 10 pair transformed to a 1,
    - a 01 pair transformed to a 0, and
    - 00 and 11 pairs are discarded $\to (1-p)^2, p^2$ of probability
  - The resulting sequence is both unbiased and uncorrelated

Mar-25 Random Generators 11

11

TRBG – Software-based: the ones in our computers, typically implemented at OS level

- Processes
  - the system clock
  - elapsed time between keystrokes or mouse movement
  - content of input/output buffers [NETWORK BUFFERS]
  - user input
  - operating system values such as system load and network statistics

Note: they are not so random: adversary that sends to me certain traffic that fills your buffer in a certain way so attacker can make educated guesses on what bits you can produce.

Mar-25 Random Generators 12

12

TRBG – Software-based

- Subject to observation and manipulation
- Use as many sources of randomness as possible
  - Mixing functions
    - E.g., Cryptographically secure hash functions (SHA-1, MD5)

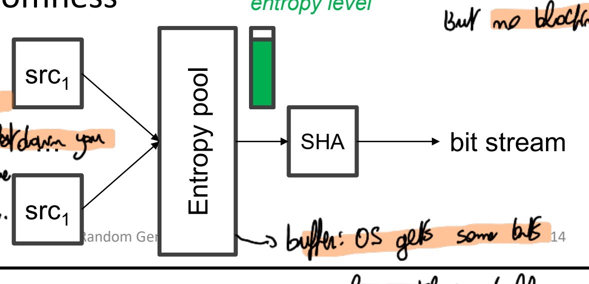
Sw-based RNG: the Linux case

- Src_i: i-th source of randomness
  - Inter-key press timing, inter-interrupt timing,…
- Two char devices
  - /dev/random: higher-quality, blocking
  - /dev/urandom: lower quality, not-blocking

2 devices: you can read from them bit streams
if you have consumed entropy
the stream will be not so random.
But no blocking

- Boot time randomness

Not random on boot for entropy pool! So at shutdown you save the value of entropy pool.



buffer: OS gets some bits from those buffers and content is given as input to a hash function.

Random Bit Generators

PSEUDORANDOM BIT GENERATORS

Mar-25 Random Generators 15

15

Pseudo Random Bit Generator

UNIVERSITÀ DI PISA

- DEFINITION. A Pseudo Random Bit Generator is a *deterministic* algorithm that, given a truly random binary sequence of lenght k (*seed*), outputs a binary sequence of length L (pseudorandom bit sequence), L >> k
  - The number of possible sequences is at most 2^k, i.e., a fraction $2^k/2^L$ of all possible sequences

Mar-25 Random Generators 16

16

Pseudo Random Bit Generator

- SECURITY INTUITION. A "small" seed is expanded into a "large" *pseudorandom* sequence in such a way that an adversary cannot "efficiently" distinguish between outputs of a PRBG and outputs of a TRG
- MINIMUM SECURITY REQUIREMENT. The length k of the seed is sufficiently large so that it is "infeasible" to search over 2^k possible output sequences (necessary condition)

Mar-25 Random Generators 17

17

Formalization

For all attackers (tests) $\mathcal{A}$, there is negligible function ε(n), s.t.:

$$\left| P_{x \leftarrow U_k}\left[A\big(G(x)\big) = 1\right] - P_{y \leftarrow U_{\ell(k)}}\left[A(y) = 1\right] \right| \leq \varepsilon(n)$$

where G: $\{0, 1\}^k \rightarrow \{0, 1\}^{\ell(k)}$, x is uniformly sampled from $\{0, 1\}^k$, U_k is the uniform distribution on k-bit strings and $U_{\ell(k)}$ is the uniform distribution on $\ell(k)$-bit strings

Mar-25 Random Generators 18

18

PRBG

- Typically
  - s_0 = seed ex. 64bits #
  - $s_{i+1} = f(s_i)$, i = 0, 1, 2,… f = generator core
- A generalization
  - s_0 = seed
  - $s_{i+1} = f(s_i, s_{i-1}, s_{i-2},…, s_{i-t},)$ generate the next value from the previous one or a combination

Mar-25 Random Generators 19

19

PRBG

- Linear Congruential Generator
  - A popular example largely used in simulation and testing
  - Definition
    - s_0 = seed
    - $s_{i+1} = (a \cdot s_i + b) \bmod m$, i = 0, 1, 2,…
    - where a, b, m are integer constants

    Not suitable for cryptography
    Because it is linear
  - ANSI C rand()
    - s[0] = 12345;
    - s[i] = 1103515245 s[i-1] + 12345 $\times 2^{31}$

Mar-25 Random Generators 20

20

LCG predictabily

- Assume a prefix s_r, s_{r+1}, s_{r+2} is known
- Define
  - $s_{r+2} = a \cdot s_{r+1} + b \mod m$
  - $s_{r+1} = a \cdot s_r + b \mod m$
  - which is a linear system of two linear equations in two unknowns (a and b) that can be "easily" solved

Mar-25 Random Generators 21

21

PRBG

- Linear Congruential Generator has good statistical properties
  - Output approximates a sequence of true random bits
  - It passes a variety of statisthical tests
- However, it is not suitable for cryptography because it is predictable need just a couple of outputs to predict behavior

Mar-25 Random Generators 22

22

Random Bit Generator

CRYPTOGRAPHICALLY SECURE PSEUDORANDOM BIT GENERATOR

CSPRBG

UNIVERSITÀ DI PISA

- Informally, a CSPRNG is an unpredictable PRNG
  - The need for unpredictability is unique for cryptography
- Informally,
  - Given a sequence of bits s_i, s_{i+1}, …, s_{i+n-1} (a prefix), for some integer n, it is «difficult» to compute the subsequent bits s_{i+n}, s_{i+n+1}, … (or any preceding bits s_{i-1}, s_{i-2}, …)
- More formally
  - Given a sequence of bits s_i, s_{i+1}, …, s_{i+n-1} (a prefix), there exist no polynomial time algorithm that can predict the next bit s_{i+n} with better than 50% chance of success

CRPRBG

- General Security Requirements
  - A PRBG is said to pass all polynomial-time statistical tests if no polynomial-time algorithm can correctly distinguish between an output sequence of the generator and a truly random sequence of the same length with probability significantly greater than 0.5
  - A PRBG is said to pass the next-bit test if there is no polynomial-time algorithm which, on input of the first t-bits of an output sequence s, can predict the (t + 1)-st bit of s with probability significantly greater than 0.5
  - Polynomial-time statistical tests and next-bit test are equivalent

Mar-25 Random Generators 26

26

CRPRBG

- CSPRBG DEFINITION. A PRBG that passes(*) the next-bit test is called cryptographically secure pseudorandom bit generator
  - (*) possibly under some plausible but unproven mathematical assumption such as the intractability of factoring integers

Mar-25 Random Generators 27

27

CSPRBG

UNIVERSITÀ DI PISA

- Ad-hoc methods, based on one-way functions
  - Hash functions, block ciphers Very flexible components, you can use them to generate random numbers (pseudo) cryptographically secure.
  - ANSI X9.17, FIPS 186
  - They have not been proven to be CSPRBG, however they are sufficient for most applications
- Based on presumed intractability of number-theoretic problem
  - RSA PRBG (integer factorization)
  - Blum-Blum-Shub PRBG (integer factorization) Generators based on PKE.

CSPRBG exist. For the most cases you can use ad-hoc methods.

Mar-25 Random Generators 28

28

Random Bit Generators

STATISTICAL TESTS

Mar-25 Random Generators 29

29

Statistical tests

UNIVERSITÀ DI PISA

- A set of statistical tests have been devised to measure the quality of an RBG
  - It is not possible to prove whether a generator is indeed an RBG; tests detect weaknesses
  - Tests provide necessary conditions → if gen fails, throw it away
    if not, no assurance it is good
    - Each test operates on a given output sequence and probabilistically determines whether it possesses a certain attribute that a truly random sequence would exhibit
  - A generator may be either rejected or accepted (i.e., not rejected)

Mar-25 Random Generators 30

30

Statistical tests

UNIVERSITÀ DI PISA

- Five basic tests
  - Frequency test (monobit test).
    - Determine whether the number of 0's and 1's are approximately the same
  - Serial test (two-bit test).
    - Determine whether the number of occurrences of 00, 01, 10, 11 are approximately the same
  - Poker test.
    - Determine whether the sequences of length *m* each appear approximately the same number of times

%

Mar-25 Random Generators 31

31

Statistical tests

- Basic tests
  - Runs test.
    - Determine whether the number of runs of various length is as expected for a random sequence
  - Autocorrelation test.
    - Check correlations between the sequence and shifted (non-cyclic) versions of it

Mar-25 Random Generators 32

32

Statistical tests

- Maurer's universal statistical test
  - Intuition: It is not possible to significantly compress (without loss of information) the output sequence of a random generator
  - Determine a very general class of possible defects (universality)
    - Including defects detectable by basic tests
  - Require a longer sequence than basic tests but more efficient than basic tests

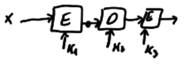
→ more general than previous ones but requires longer sequences

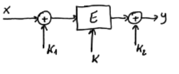
Mar-25 Random Generators 33

33

Either we resolve to TRBG (HW or SW based). If we need to generate PRBG, we cannot use simulators. We need other generators, typically ad-hoc generators. There are tests to check if gen is good, but only gives necessary conditions.

$y = E_{(K_1,K_2,K_3)}(x) = E_{K_3}(D_{K_2}(E_{K_1}(x)))$





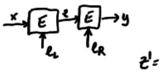
$y = E_K(x \oplus K_1) \oplus K_2$

$x = D_K(y \oplus K_2) \oplus K_1$

$\forall K_1, K_2 \in K \;\; \exists K_3 \in K \mid \forall x \in M$

$E_{K_1}(E_{K_2}(x)) = E_{K_3}(x)$

$y = 2E_{(e_L,e_R)}(x) = E_{e_R}(E_{e_L}(x))$



$z' = D_{e_R}(y)$

e_L	z
-	-
⋮	⋮
-	-

Mar-25 Random Generators 34

34