(Pseudo)random Data

Entropy, CSPRNG

Marek Sýs, syso@mail.muni.cz
Faculty of Informatics, Masaryk University

CROCS
Centre for Research on
Cryptography and Security

RNG types

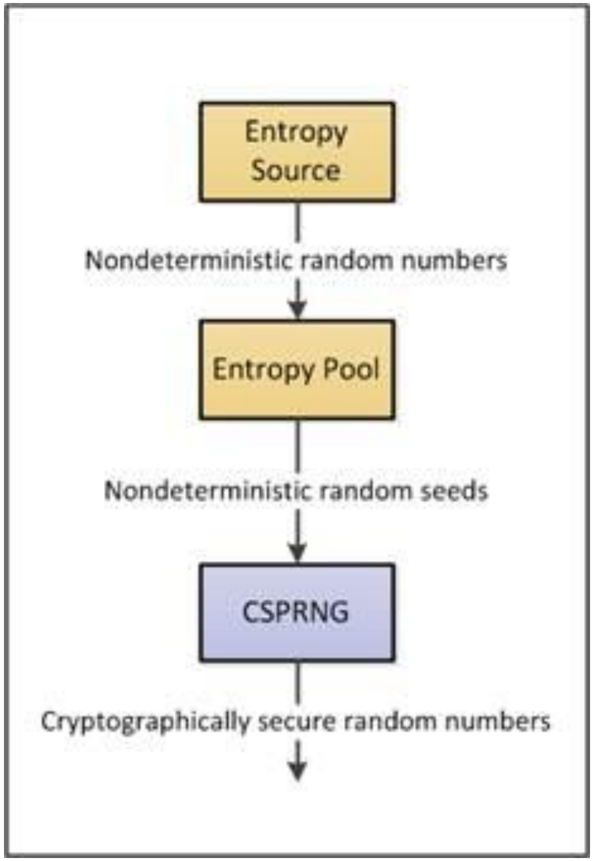
- True RNG (TRNG)
  - collects random events:
    - hardware noise, physical phenomena
    - behaviour of system/user
  - Pros: independent values, aperiodic sequence
  - Cons: often slow, often small density of entropy
- Pseudorandom RNG (PRNG)
  - software – imitates randomness
  - Pros: fast
    - CSPRNG - uniform, **practically** independent
  - Cons: deterministic, periodic, fixed entropy

Combined TRNG + (CS)PRNG

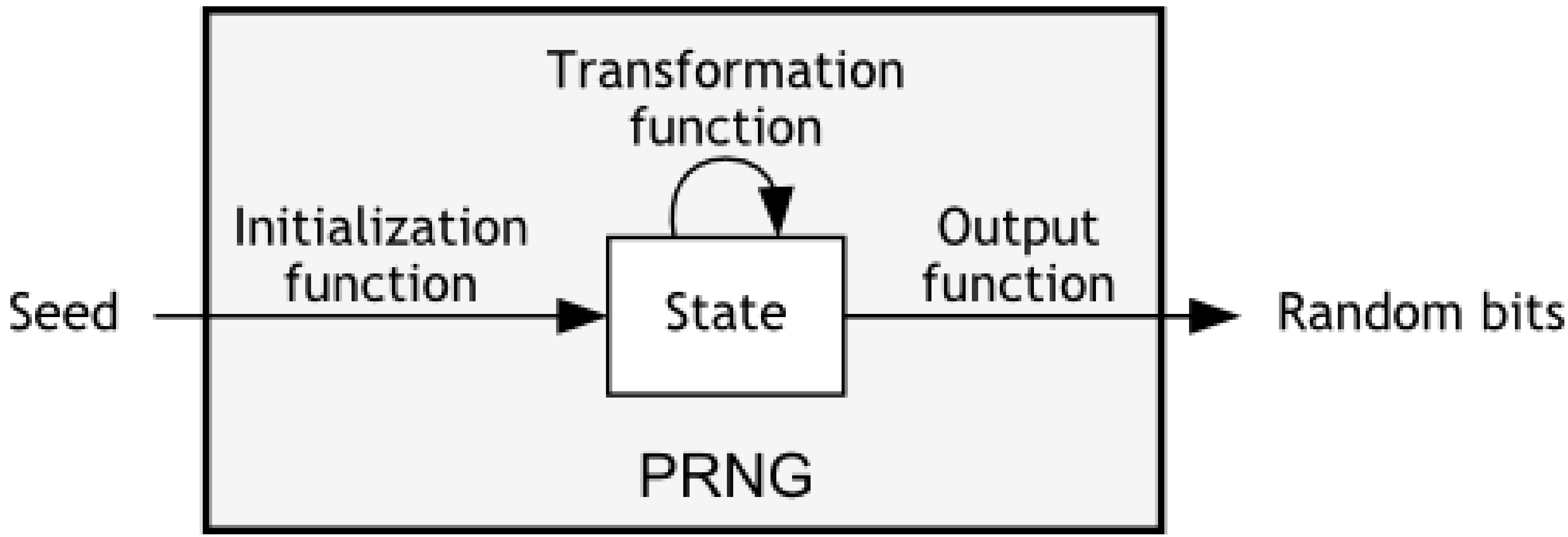
- TRNG (entropy source)

- Entropy Pool (optional)
  - collects entropy
  - entropy is increased – fresh entropy mixed into pool

- CSPRNG
  - seeded by pool or TRNG directly



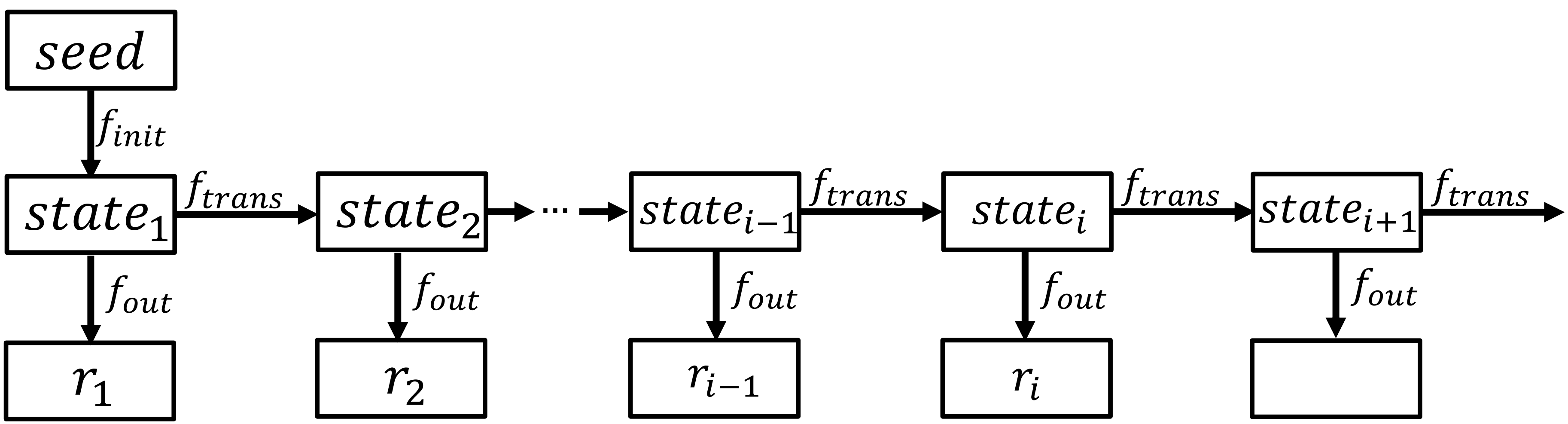
PRNG



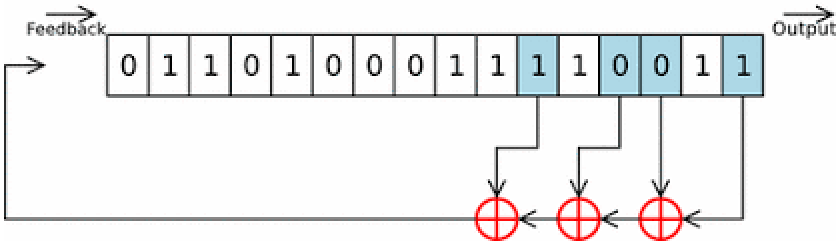
Source: https://pit-claudel.fr/clement/blog/how-random-is-pseudo-random-testing-pseudo-random-number-generators-and-measuring-randomness/

Security of PRNG

- Seed and internal state must be protected
  - must be large enough – not allowing brute force
  - seed - determines whole sequence
  - state $state_i$ determine next values $r_i, r_{i+1}, r_{i+2}, \ldots$
    - inappropriate f_{trans} also previous values $r_1, r_2, \ldots, r_{i-1}$



Insecure PRNGs



- LFSR of n-bits
  - Output bit - linear function state bits
  - Berlekamp Massey alg. can compute IS from 2n output bits
- Standard random function:
  - Linear congruential generator (LCG)
  - return whole or half of IS
- RC4 – P(2nd byte =0)=1/128 (not 1/256)
- Mersenne twister – IS from 624 values
- Anything not labeled as CSPRNG…

LCG

- gcc random(), java.util.Random, …

$$state_{i+1} = a.state_i + c \bmod m$$

Source	modulus *m*	multiplier *a*	increment *c*	output bits of seed in *rand()* or *Random(L)*
ZX81, *ZX Spectrum* [18]	$2^{16} + 1$	75	0	$(x_n - 1) / 2^{16}$
Numerical Recipes `ranqd1`, Chapter 7.1, §An Even Quicker Generator, Eq. 7.1.6 parameters from Knuth and H. W. Lewis	2^{32}	1664525	1013904223	
Borland C/C++	2^{31}	22695477	1	bits 30..16 in *rand()*, 30..0 in *lrand()*
glibc (used by GCC)[19]	2^{31}	1103515245	12345	bits 30..0

- significant part of $state_i$ returned as random value r_i
- linear state update func. - states can be **backtracked**!

Randomness and entropy

- What is random, data or RNG? Is 4 random?
  - randomness of data determined by RNG
- Entropy is **measure** of randomness
  - expressed in bits
  - better randomness ⇒ larger entropy
- Entropy represents:
  - uncertainty of produced values and
  - average amount of information values carry and
  - attack complexity on key / security of system.
  - see other definitions by NIST

Entropy computation

- **Assumption**: RNG produces independent values.

- Defined for RNG output represented by random variable X.
- (Shannon) entropy: $$\mathcal{H}(X) = -\sum_{i=1}^{n} p_i(X = x_i) \log p_i(X = x_i)$$

- Property: same entropy when modeled RNG differently
  - single bits $$p_0 = 0.3, p_1 = 0.7 \implies \mathcal{H}(X) = 0.88$$
  - 2-bit blocks: $$p_{00} = 0.09, p_{01} = p_{10} = 0.21, p_{11} = 0.49 \implies \mathcal{H}(X) = 2 * 0.88$$

- For impossible events: $$p_i(x_i) = 0 \implies p_i(x_i) \log p_i(x_i) = 0$$

Entropy estimate - example

- Source (https://qrng.anu.edu.au/RainBin.php):
- Estimates for sequence of 80 bits:

`0110110101010011000001011000000101101000`
`0100111111101110000010101100010001111111`

  - Model (1bit): 40x 0, 40x 1 – entropy $\mathcal{H}(X) = 1$
  - Model (2bit): 12x'01', 6x'10', 11x'00', 11x'11' – $\mathcal{H}(X) = 1.95$
- Estimates for two 960-bit seqs for 1-6 bit model:
  - 0.9999, 0.9992, 0.9993, 0.9910, 0.9704, 0.9470
  - 0.9998, 0.9974, 0.9913, 0.9849, 0.9734, 0.9485

Entropy pool - mixing entropy

- Operations: replacement : XOR : Hash:
  - replacement – can decrease the entropy!
  - XOR – cannot decrease entropy
    - but may not increase if part already full!
  - hashing – increases (up to hash size) and spreads entropy