Simulações de Monte Carlo e geração de números aleatórios

Prof. Renato Pimentel

2025/2

Sumário

1. Geração de números aleatórios
   - Monte Carlo
   - Aleatoriedade
   - Números pseudo-aleatórios
   - Alguns geradores de números aleatórios
   - Qualidade dos geradores
   - Semente aleatória (*seed*)

Monte Carlo

- Um **método de Monte Carlo** ou **simulação de Monte Carlo** é um tipo de algoritmo computacional que usa repetições de geração de números aleatórios para a obtenção de um resultado numérico.
- O nome é uma referência ao **Casino de Monte Carlo**, complexo de jogos e entretenimento fundado em 1865 na localidade de mesmo nome, no Principado de Mônaco.
  - **Stanislaw Ulan** (1909-84), matemático polonês que desenvolveu o método, adotou o nome inspirado nos hábitos de apostador de um tio.

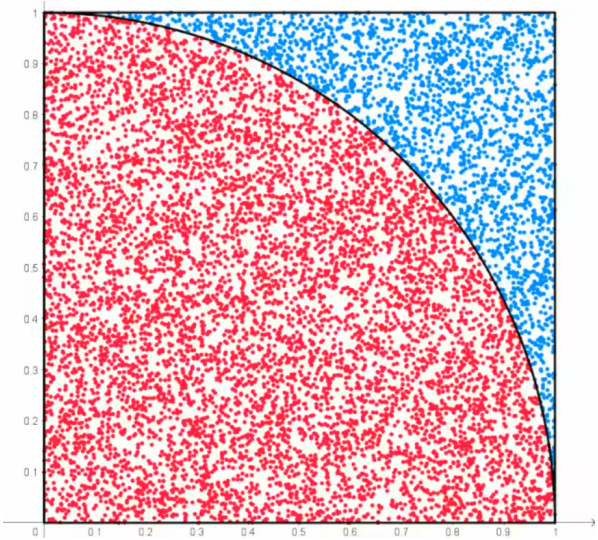
Fonte: https://commons.wikimedia.org/w/index.php?curid=54618166

Forma geral

1. Defina o domínio dos possíveis resultados (espaço amostral);
2. Gere números aleatoriamente a partir de uma função distribuição de probabilidade;
3. Calcule os resultados de forma determinística, a partir dos valores;
4. Combine os resultados ao final;

Exemplo clássico: estimativa de π

- Imagine um quadrado de lado 1, e um quadrante inscrito no mesmo;
- Distribua uniformemente um conjunto de pontos sobre o quadrado;
- Conte o número de pontos dentro do quadrante (i.e. cuja distância da origem é menor que 1);
- A razão entre o número de pontos dentro do quadrante e o total de pontos distribuídos **tende** a $\pi/4$. Multiplique por 4 e obtenha a estimativa de π.



Fonte: adaptado de https://commons.wikimedia.org/w/index.php?curid=140013480

Implementação em Python

```
import numpy as np
rng = np.random.default_rng()

N = 100_000
n = 0

x = rng.uniform(size=N)
y = rng.uniform(size=N)

for i in range(N):
    if x[i]**2 + y[i]**2 < 1:
        n+=1

pi = 4*n/N
```

Adaptado de: https://pt.wikipedia.org/wiki/M%C3%A9todo_de_Monte_Carlo

Outros exemplos comuns

- Lançamento de um dado justo: quais as chances de cada resultado (distribuição)? Qual o resultado esperado?
- Lançamentos de uma moeda;
- Simulação de sorteios;
- Cálculo de uma integral definida de uma função conhecida;
- **Processos estocásticos**.

Exercícios

1. Suponha que temos uma caixa com 5 bolas pretas e 7 bolas brancas. Compramos duas sem colocar de volta. Qual é a distribuição de probabilidade da variável aleatória $X =$ número de bolas pretas retiradas? E qual é o número esperado de bolas pretas retiradas?
2. No RPG *Dungeons & Dragons*, para superar um desafio, o jogador deve lançar um dado de 20 lados ($d20$) e comparar o resultado a um número alvo. Em certas situações, o jogador ganha um benefício chamado **vantagem**, e pode jogar dois $d20$ e escolher o maior valor. Qual o valor esperado de um lançamento sem e com vantagem?

Sumário

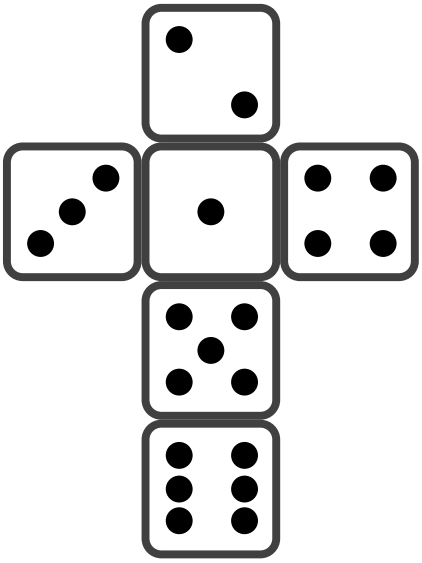
1. Geração de números aleatórios
   - Monte Carlo
   - Aleatoriedade
   - Números pseudo-aleatórios
   - Alguns geradores de números aleatórios
   - Qualidade dos geradores
   - Semente aleatória (*seed*)

Recapitulando...

O que é aleatório?

- No universo: pergunta filosófica;
- Na computação: **nada**;
- O que fazemos é **simular** aleatoriedade.

Como?



"The randomness is only apparent; it is a result of ignorance."
– Laplace

"Random numbers should not be generated with a method chosen at random."
– Knuth

"Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin."
– Von Neumann

Sumário

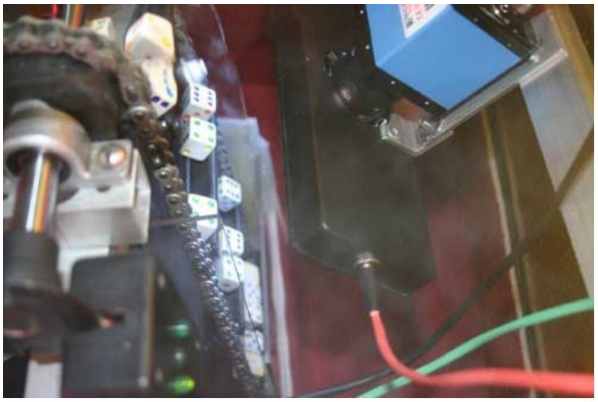
1. Geração de números aleatórios
   - Monte Carlo
   - Aleatoriedade
   - Números pseudo-aleatórios
   - Alguns geradores de números aleatórios
   - Qualidade dos geradores
   - Semente aleatória (*seed*)

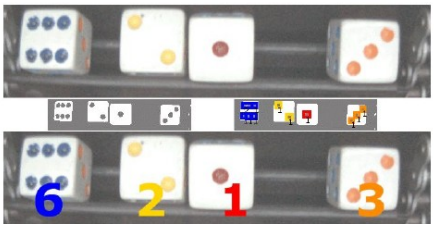
Números pseudo-aleatórios

Um bom gerador é necessário para:

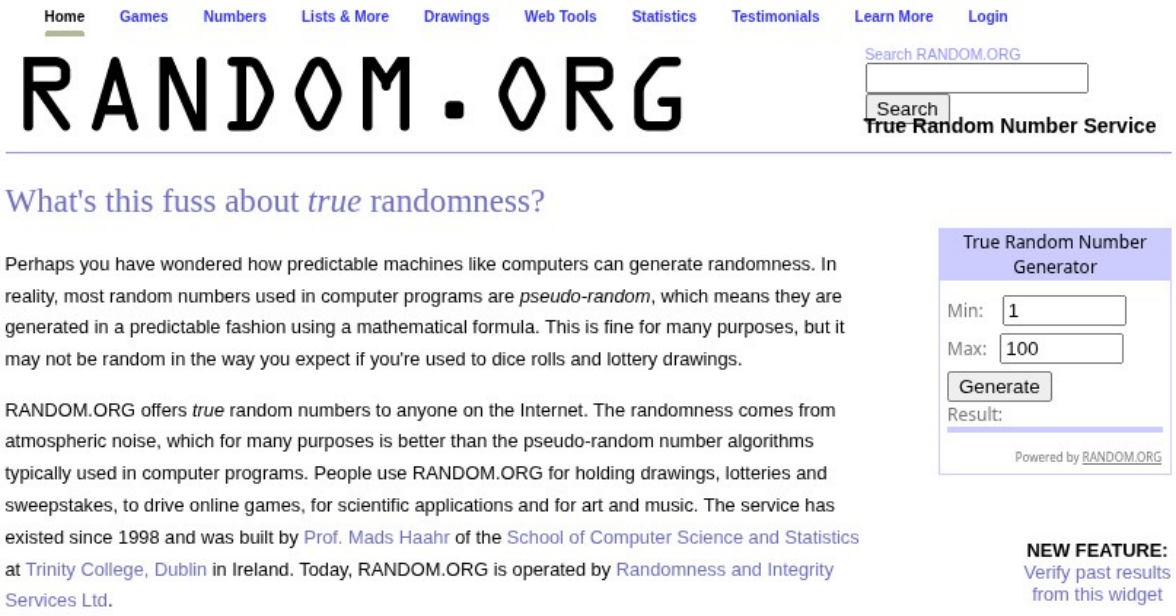
- Resultados confiáveis no método de Monte Carlo;
- Criptografia;
- Jogos.

Solução: Dice-O-Matic?
https://www.youtube.com/watch?v=7n8LNxGbZbs





https://www.random.org/



- Oferece números verdadeiramente aleatórios;
- A aleatoriedade vem do ruído atmosférico;
- Melhor que algoritmos pseudo-aleatórios para muitos propósitos.

Lista de geradores:
https://en.wikipedia.org/wiki/List_of_random_number_generators

Sumário

1 Geração de números aleatórios
- Monte Carlo
- Aleatoriedade
- Números pseudo-aleatórios
- Alguns geradores de números aleatórios
- Qualidade dos geradores
- Semente aleatória (*seed*)

Soluções Clássicas

Método do meio do quadrado (Von Neumann)

1. Comece com qualquer número de quatro dígitos como **semente**;
2. Eleve ao quadrado para obter um número de oito dígitos;
3. Pegue os quatro dígitos do meio como próximo número aleatório;
4. Repita o quanto necessário.

Problema

Bem simples! Mas ruim!

> *"Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin."*
> *– John Von Neumann (1951)*

Soluções Clássicas (Melhor)

Linear Congruential Generator (LCG)

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

- Pode gerar bons resultados ("eficiência"?);
- Depende muito dos parâmetros m e a.

IBM PCs antigos (RANDU)

$$a = 65539$$
$$c = 0$$
$$m = 2^{31} = 2147483648$$

Parâmetros de LCGs Comuns

Fonte	módulo m	multiplicador a	incremento c
ZX81	$2^{16}+1$	75	74
Numerical Recipes	2^{32}	1664525	1013904223
Borland C/C++	2^{32}	22695477	1
glibc (GCC)	2^{31}	1103515245	12345
ANSI C	2^{31}	1103515245	12345

Solução Moderna (Melhor Ainda)

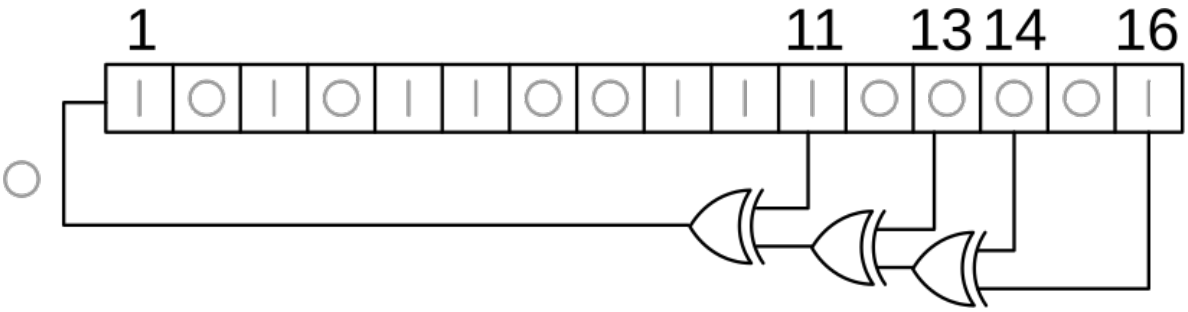
Mersenne Twister (1997)

- Possivelmente o mais usado hoje em dia;
- Passa mais testes de aleatoriedade que o LCG;
- Baseado no **número primo de Mersenne** dado por $2^{19937}-1$;
- Período longo! ($\approx 10^{6000}$);
- Ainda não é criptograficamente seguro!
  - É possível fazer engenharia reversa do estado interno do gerador observando algumas centenas de saídas, e a partir daí prever todos os resultados futuros.

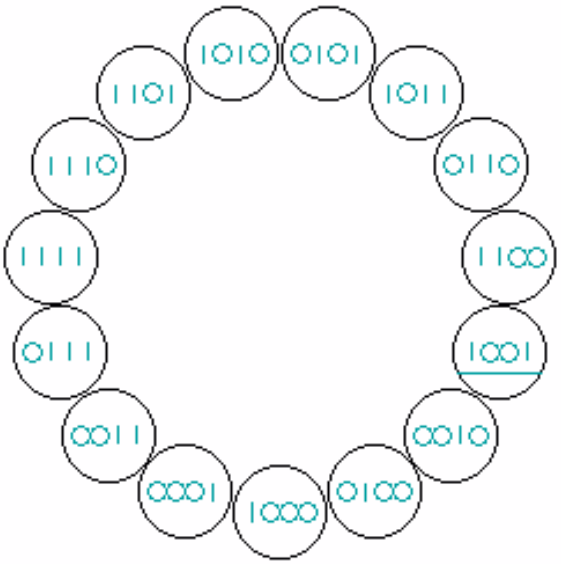
Solução Moderna

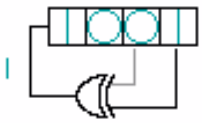
Linear Feedback Shift Register (LFSR)

- Baseado em deslocamento de números binários;
- Período **muito** grande;
- https://en.wikipedia.org/wiki/Linear-feedback_shift_register



Exemplo de sequência LFSR





Python

- Módulo nativo random: Mersenne Twister MT19937.
- Módulo numpy.random: PCG64[1] (*permuted congruential generator*, versão aprimorada do LCG, com 64 bits).
  - Invocado com a instrução rng = numpy.random.default_rng().
  - Discussão: https://github.com/numpy/numpy/issues/13635
- Módulo nativo secrets (a partir da versão 3.6): Usa RNG do sistema operacional, foca na **segurança** (números aleatórios criptograficamente fortes).

[1] https://www.pcg-random.org/

Ajustando o range

Como ajustar o ***range*** dos números gerados?

Exemplo em Python

```
np.random.random()  # retorna float entre 0 e 1
```

Fonte: Domínio Público, https://commons.wikimedia.org/w/index.php?curid=119900

Método da transformação inversa

- Se U é v. a. distribuída uniformemente no intervalo $(0, 1)$, então para qualquer função de distribuição contínua F, é possível definir v. a. X tal que
$$X = F^{-1}(U)\,.$$
- Temos que $F_X(a) = P(X \leq a)$.

Exemplo: exponencial

- Considere $F(x) = 1 - \mathrm{e}^{-\lambda x}, x \geq 0$.
- Então, $F^{-1}(u)$ é tal que $1 - \mathrm{e}^{-\lambda x} = u$.
- Isolando-se então x, $x = -\frac{1}{\lambda}\ln(1-u)$, então se U é v. a. uniforme em $(0, 1)$, temos que
$$F^{-1}(U) = -\frac{1}{\lambda}\ln(1-U)$$
é exponencialmente distribuída com média $1/\lambda$.

Caso discreto

- Se o objetivo é simular uma v. a. X tal que
$$P(X_j) = P_j, \quad j = 1, 2, \ldots, \qquad \sum_j P_j = 1\,,$$
- Pode-se fazer um análogo discreto do método anterior, considerando U é v. a. distribuída uniformemente no intervalo $(0, 1)$:
$$X = \begin{cases} x_1, & \text{se } U < P_1, \\ x_2, & \text{se } P_1 \leq U < P_1 + P_2, \\ \vdots & \\ x_j, & \text{se } \sum_1^{j-1} P_i \leq U < \sum_1^{j} P_i, \\ \vdots & \end{cases}$$
- Veja que $P(X = x_j) = P\left\{\sum_1^{j-1} P_i \leq U < \sum_1^{j} P_i\right\} = P_j$.

Sumário

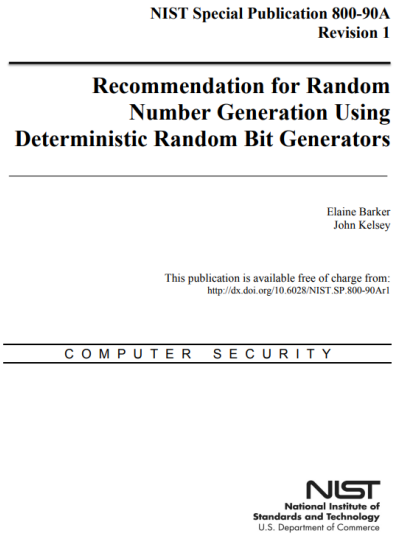
1. Geração de números aleatórios
   - Monte Carlo
   - Aleatoriedade
   - Números pseudo-aleatórios
   - Alguns geradores de números aleatórios
   - Qualidade dos geradores
   - Semente aleatória (*seed*)

Teste de Aleatoriedade

Como avaliar um bom gerador de números aleatórios?

Problemas:

- A longo prazo, padrões emergem;
- Possível período, resultados repetem;
- NIST[2]: **testes estatísticos**.


NIST Special Publication 800-90A
Revision 1

Recommendation for Random Number Generation Using Deterministic Random Bit Generators

Elaine Barker
John Kelsey

This publication is available free of charge from:
http://dx.doi.org/10.6028/NIST.SP.800-90Ar1

COMPUTER SECURITY

NIST
National Institute of Standards and Technology
U.S. Department of Commerce

https://doi.org/10.6028/NIST.SP.800-90Ar1

[2] *National Institute of Standards and Technology*

Propriedades de Bom Gerador

- Uniformidade;
- Independência;
- Imprevisibilidade;
- Período Longo.

Exemplo (humorístico)

```
def getRandomNumber():
    return 4 # escolhido de rolagem de dado justo,
             # garantido ser aleatorio
```


Teste de Aleatoriedade *DieHard* I

Os **testes *DieHard*** – `https://ani.stat.fsu.edu/diehard/` – são uma bateria de testes estatísticos usados para medir a qualidade de um gerador de números aleatórios. Eles foram desenvolvidos por George Marsaglia ao longo de vários anos e publicados pela primeira vez em 1995, em um CD-ROM de números aleatórios.
Testes:

- Intervalos de aniversários (*Birthday spacings*): Escolhem-se valores aleatórios em um grande intervalo. Os espaçamentos entre os valores devem ser assintoticamente distribuídos de forma exponencial. O nome vem do paradoxo do aniversário.
- Permutações sobrepostas (*Overlapping permutations*): Analisam-se sequências de cinco números aleatórios consecutivos. As 120 ordens possíveis devem ocorrer com probabilidade estatisticamente igual.

Teste de Aleatoriedade *DieHard* II

- Postos de matrizes (*Ranks of matrices*): Selecionam-se alguns bits de vários números aleatórios para formar uma matriz sobre $\{0, 1\}$, e então determina-se seu posto. Contam-se as frequências dos postos obtidos.
- Testes do macaco (*Monkey tests*): Sequências de um certo número de bits são tratadas como "palavras". Contam-se as palavras sobrepostas em um fluxo. O número de "palavras" que não aparecem deve seguir uma distribuição conhecida. O nome vem do **teorema do macaco infinito**.
- Contagem dos 1s (*Count the 1s*): Conta-se o número de bits iguais a 1 em cada byte sucessivo (ou em bytes escolhidos). As contagens são convertidas em "letras" e depois contam-se as ocorrências de "palavras" de cinco letras.

Teste de Aleatoriedade *DieHard* III

- Teste do estacionamento (*Parking lot test*): Colocam-se aleatoriamente círculos unitários em um quadrado de 100×100. Um círculo é estacionado com sucesso se não sobrepõe outro já estacionado. Após 12.000 tentativas, o número de círculos estacionados com sucesso deve seguir uma certa distribuição normal.
- Teste da distância mínima (*Minimum distance test*): Colocam-se aleatoriamente oito mil pontos em um quadrado de 10000×10000, e calcula-se a menor distância entre pares de pontos. O quadrado dessa distância deve ser distribuído exponencialmente com uma certa média.
- Teste das esferas aleatórias (*Random spheres test*): Escolhem-se aleatoriamente quatro mil pontos em um cubo de aresta mil. Centra-se uma esfera em cada ponto, cujo raio é a distância mínima até outro ponto. O volume da menor esfera deve ser distribuído exponencialmente com uma certa média.

Teste de Aleatoriedade *DieHard* IV

- Teste do aperto (*The squeeze test*): Multiplica-se 231 por números reais aleatórios no intervalo $(0, 1)$ até alcançar 1. Repete-se isso 100.000 vezes. O número de multiplicações necessárias deve seguir uma certa distribuição.
- Teste das somas sobrepostas (*Overlapping sums test*): Gera-se uma longa sequência de números reais aleatórios no intervalo $(0, 1)$. Somam-se blocos de cem números consecutivos. As somas devem ser distribuídas normalmente, com média e variância características.
- Teste das sequências ascendentes/descendentes (*Runs test*): Gera-se uma longa sequência de números reais aleatórios no intervalo $(0, 1)$. Contam-se as subsequências ascendentes e descendentes. As contagens devem seguir uma certa distribuição.
- Teste do craps (*The craps test*): Jogam-se 200.000 partidas de craps (par de dados), contando as vitórias e o número de lançamentos por jogo. Cada contagem deve seguir uma certa distribuição.

Teste mais moderno: TestU01

http://simul.iro.umontreal.ca/testu01/tu01.html

- **Chave:** equidistribuição multidimensional

ARTICLE

TestU01: A C library for empirical testing of random number generators

Authors: Pierre L'Ecuyer, Richard Simard | Authors Info & Claims

ACM Transactions on Mathematical Software (TOMS), Volume 33, Issue 4 • Article No.: 22, Pages 1 - 40
https://doi.org/10.1145/1268776.1268777

Published: 15 August 2007 Publication History | Check for updates

https://doi.org/10.1145/1268776.1268777

Sumário

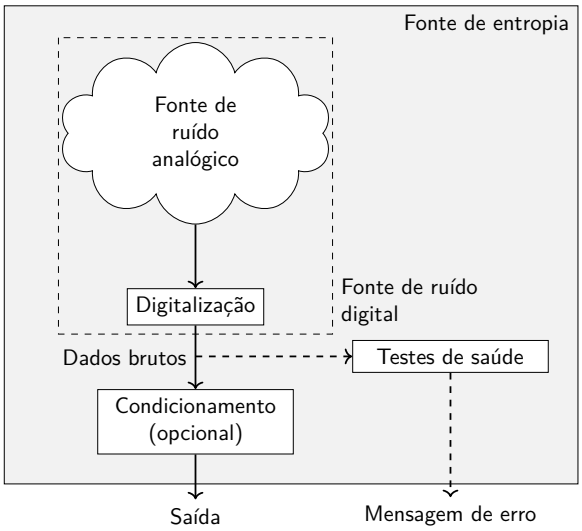
1. Geração de números aleatórios
   - Monte Carlo
   - Aleatoriedade
   - Números pseudo-aleatórios
   - Alguns geradores de números aleatórios
   - Qualidade dos geradores
   - Semente aleatória (*seed*)

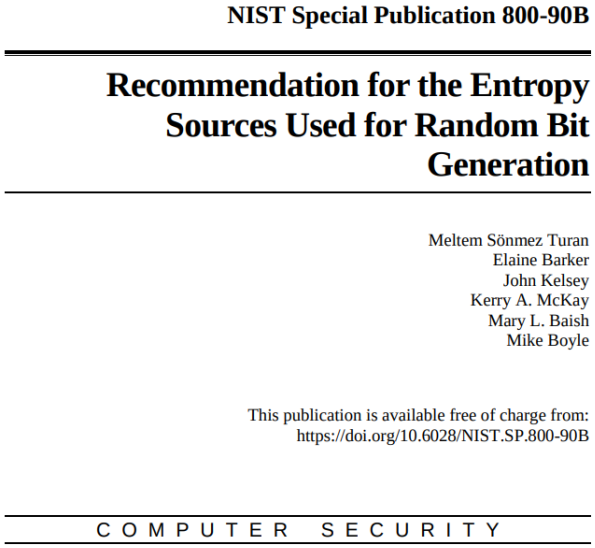
De onde vem a *seed*?

Entrada **manual**? Por quê?

Melhor: fontes de entropia

- Física (hardware)
- Não-física (RAM, *system time*, movimentos do mouse, etc.)




NIST Special Publication 800-90B

Recommendation for the Entropy Sources Used for Random Bit Generation

Meltem Sönmez Turan
Elaine Barker
John Kelsey
Kerry A. McKay
Mary L. Baish
Mike Boyle

This publication is available free of charge from:
https://doi.org/10.6028/NIST.SP.800-90B

COMPUTER SECURITY

https://doi.org/10.6028/NIST.SP.800-90B

Geradores seguros para criptografia

- ChaCha20 (geral) – https://cr.yp.to/chacha.html
- Cryptography API (Windows) – https://learn.microsoft.com/en-us/windows/win32/seccng/cng-portal
- Urandom (Linux)

Conceito essencial: Atualizar entropia!

Fontes de entropia:

- Movimentos do mouse;
- Teclas e *timings* de teclado;
- Tempos de *threads*;
- Estatísticas de sistemas;
- Conteúdos de memória.

Referências

1. ROSS, S. *Introduction to Probability Models*, 13. ed. (Academic Press), 2024.
2. https://www.probabilitycourse.com/

Os materiais de parte desta seção foram gentilmente cedidos por Leandro N. Couto (FACOM/UFU).
Adaptações e LaTeXagem: Renato Pimentel, FACOM/UFU