ALGORITMOS E ALEATORIEDADE EM COMPUTAÇÃO*

Jean Piton-Gonçalves

> *Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin*[1] *(John Von Neumann - 1951).*

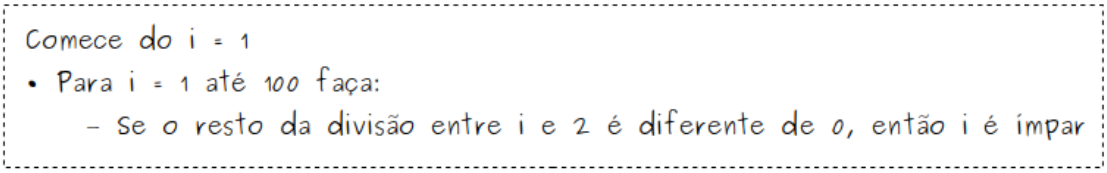
Imagine que você necessite enviar uma carta composta pelos 50 primeiros números ímpares $(1,3,5,\ldots,97,99)$ para alguém que resida em uma ilha muito distante do continente. Essa carta terá que ser informativa e resumida, pois sabe-se que o custo de envio é de R$ 1.00 por caractere. Por exemplo, a mensagem $1,3,7,11$ possui 8 caracteres (vírgula é contabilizada) e custaria R$ 8.00. Sendo assim, a carta de números ímpares é composta por 144 caracteres e custaria R$ 144.00 para ser enviada.

Todavia, o problema é que você só dispõe de R$ 70.00. Então surge a seguinte questão: "*É possível enviar a mensagem de forma mais barata, sem alterar seu mesmo conteúdo?*" SIM, é possível. Portanto, iremos analisar o conteúdo dessa mensagem e verificar se existe outra forma de exprimi-la ou interpretá-la.

Uma vez que a mensagem é composta somente por números ímpares, utilizaremos a informação de que os ímpares não retornam uma divisão inteira por 2 (por exemplo, $5 \div 2 = 2.5$ ou $81 \div 2 = 40.5$). A partir dessa informação, conseguiremos decidir se, dado um número inteiro positivo, ele é ou não divisível por 2.

E se simplesmente enviarmos a "receita" para gerar os 50 ímpares? Essa "receita" ou sequência de passos é um algoritmo[2] que pode ser desenvolvido apenas com lápis e papel e, nessa direção, o algoritmo em língua nativa (portuguesa) que gera a sequência dos 50 primeiros ímpares é escrito na Figura 1.1. Observe que a divisão é verificada 100 vezes, caracterizando um laço[3].

Figura 1.1: Algoritmo em língua nativa que gera uma sequência de pares.



Fonte: o autor.

* DOI - 10.29388/978-65-86678-51-2-0-f.175-198

[1] Fonte: MacHale (1993).

[2] É um procedimento que envolve objetos matemáticos em um número finito de etapas, que pode envolver repetição de etapas e/ou estruturas lógicas. Um exemplo é o Algoritmo de Euclides que provém do seguinte Teorema da Divisão Euclidiana: sejam a e b números naturais com $b \neq 0$. Existem dois únicos números naturais q e r tais que $a = b \times q + r$, onde $0 \leq r < b$. Detalhes são encontrados em Cohn (1980).

[3] Em inglês, *loop*.

Nossa mensagem passará a ser o próprio algoritmo que soma 66 caracteres[4], diminuindo o comprimento da mensagem e custando R$ 66.00 (sobrando ainda R$ 4.00 no bolso!). O mais interessante dessa "mensagem algorítmica" é que podemos enviar uma quantidade maior de números ímpares pagando bem pouco a mais. Por exemplo, se quiséssemos enviar 10000 números ímpares o custo seria de, apenas, R$ 68.00, uma vez que acrescentaríamos dois zeros no 100 para se tornar 10000. Importunamente, o trabalho do destinatário na geração desses ímpares seria dispendioso e, para isso, os algoritmos implementados em computador seriam uma solução.

Antes de um algoritmo ser implementado no computador, ele deve ser transformado em um pseudocódigo[5], que é extremamente importante para uma execução em um computador. Um pseudocódigo popular no Brasil é o Portugol, que é uma pseudolinguagem estruturada em língua portuguesa. O algoritmo que gera números ímpares pode ser traduzido para Portugol em um pseudocódigo, conforme mostra o Código 1.1.

Código 1.1: Pseudocódigo GeraImpar.

```
Pseudocódigo GeraImpar
        inicio
            i=1
    Para i=1 até 100 faça
        Se resto(i,2) <> 0
    então escreva(i,"é ímpar")
            fim_Se
        fim_Para
    fim
```

Fonte: o autor**.**

Para ser executado (rodar) no computador, o pseudocódigo pode deve ser traduzido para um código-fonte. Por exemplo, a implementação na Linguagem Java é mostrada no Código 1.2.

Código 1.2: Código-fonte em Java.

```
public class GeraImpar {
public static void main(String[] args) {
        for (int i = 1; i < 100; i++) {
            if ((i % 2) != 0) {
     System.out.println("Os ímpares são:" + i);
                }
            }
    }
    }
```

Fonte: o autor.

[4] Cada espaço em branco é considerado um caractere.
[5] É uma forma de expressar um algoritmo em uma linguagem simples ou natural e que não está diretamente relacionada à uma linguagem de programação.

Após compilar[6] e executar o Código 1.2 teremos automaticamente a lista com os 50 primeiros números ímpares. Observe que há uma estrutura lógica determinística no código, que deve respeitar a sintaxe e a semântica que cada linguagem de programação oferece.

Essa instanciação, que vai desde o algoritmo na sua forma mais primária até sua versão final, tem como objetivo motivar o leitor no que se refere à geração de números por meio de algoritmos. Para saber mais sobre o assunto, recomendamos a leitura de Cormen, Leiserson e Stein (2012).

Embora a situação do envio da mensagem seja fictícia, ela demonstra que determinados tipos de problemas podem ser traduzidos ou resolvidos por um algoritmo. Uma vez que estes são componentes de sistemas operacionais e de *softwares*, podemos dizer que tudo o que envolve equipamentos dotados de tecnologias digitais dependem de um conjunto de algoritmos bem implementados em uma ou mais linguagens de programação.

As ligações entre o acaso e o algoritmo são definidas pela geração de números aleatórios. Um Gerador de Números Aleatórios ou *Random Number Generators* (RNG) é um dispositivo de *software* ou de *hardware* que, ao ser acionado, gera uma saída numérica inesperada (GENTLE, 2004; Dutang e Würtz, 2009). Esse número gerado é classificado em:

- **Aleatório verdadeiro**. É gerado por um ou mais fenômenos físicos, em uma condição não-determinística. São exemplos, os contadores de elementos radioativos, urnas mecânicas e uma jogada de dados por um ser humano.
- **Pseudo-aleatório**. É gerado por algum procedimento que culminará em uma condição determinística, ou seja, é um número gerado por algum algoritmo determinístico.
- **Quasi-aleatório**. É gerado a partir de sequências quasi-aleatórias que, essencialmente, buscam uma distribuição o mais uniforme possível dos números gerados.
- **Aleatório imprevisível**. É gerado por fontes de entropia que se baseiam em um fenômeno imprevisível baseado na interação homem-computador.

Um conceito importante que será utilizado ao longo desse texto é o de *sequências de aleatórios*, isto é, trata-se de uma sequência finita de números aleatórios. São exemplos as sequências de inteiros: (859,601,981,846,928,413,864,911,500,546), (868,845,27,252,204) e (960,683,636,594,125). Tais sequências possuem um papel muito importante na Criptografia, pois é possível (sob certas condições) descobrir o conteúdo de uma mensagem codificada.

Números aleatórios verdadeiros

O denominado *gerador de números verdadeiramente aleatórios* ou *True Random Number Generator* (TRNG) busca obter, a partir de uma situação não-determinística, um aleatório

[6] É o processo que transforma o código-fonte em código de máquina, que é aquele interpretado pelo sistema operacional fazendo-o "rodar".

verdadeiro. O princípio do TRNG é transformar sinais de fenômenos físicos não determinísticos em um número aleatório a partir de um dispositivo físico que se conecta à um computador e/ou equipamento digital. Os dispositivos TRNGs são aplicados, por exemplo, na geração de chaves na criptografia, sorteio em loterias, simulação e experimentação em estatística, equipamentos militares (tais como mísseis e aviões de combate).

A Entropia[7] ligada à computação torna-se uma importante medida para o nível de aleatoriedade. A verdadeira aleatoriedade é bastante difícil de alcançar, uma vez que as CPUs são, por natureza, de funcionamento determinístico. Portanto, os geradores aleatórios devem buscar a entropia em outras fontes, que podem ser a partir do recolhimento de informações de diversos dispositivos periféricos, tais como falhas na memória RAM, mudanças de contexto da CPU, etc.

Em meados dos anos 2000, Davies (2000) discutiu sobre o funcionamento e a importância dos dispositivos TRNGs, sendo um deles (ver Figura 1.2) desenvolvido pela empresa sueca Protego[8] que, quando ligado a um computador por meio da porta serial (*serial bus*), gerava uma sequência de aleatórios verdadeiros.

Figura 1.2: Dispositivo NVA da Protego em 2000.

Fonte: Davies (2000).

[7] Na computação a Entropia é a aleatoriedade coletada por fontes de hardware para um sistema operacional ou software. Um exemplo é o movimento do mouse.

[8] Desde 1995 eles vêm desenvolvendo soluções para a geração de NVAs que são diretamente aplicados em loterias e jogos e mais informações são obtidas no site oficial da empresa: https://www.protegost.com

Esse tipo de dispositivo existe até hoje, e vem sendo atualizado constantemente, permitindo-se utilizar em uma simples porta USB, como é mostrado na Figura 1.3.

Figura 1.3: Dispositivo NVA da Protego em 2020.

Fonte: Repositório de imagens do *site* da Protego[9]

Os dispositivos TRNG utilizam transdutores[10] que, quando associados à um conversor analógico/digital, geram *bits* e, em seguida, reúne-os em *bytes*. No caso específico de *hardwares*, o circuito eletrônico converte o ruído elétrico em um número aleatório.

Com destaque às aplicações de tais dispositivos, na indústria automobilística, o *E-Safety Vehicle Instrusion Protected Applications Hardware Security Module* (EVITA HSM) atua na criptografia avançada presente na cybersegurança de veículos automotivos, principalmente em redes de comunicação dos sensores (do ABS, AirBag, etc) (SJAFRIE, 2019). De acordo com o autor, existe aplicação direta de TRNG em veículos autônomos, tanto na parte de sensores, como da segurança contra a invasão de hackers por ondas de rádio.

Na área da saúde, o trabalho de Camara *et al.* (2019) menciona que equipamentos médicos ou dispositivos de uso geral por pacientes são dotados de sensores que podem ter conectividade sem fio. Se a transmissão de dados sensíveis é conduzida por meio de um canal de rádio inseguro, então há a necessidade de se garantir mecanismos seguros (criptografia) nessa transmissão, necessitando de um gerador de TRNG rápido e robusto.

[9] Disponível em https://www.protegost.com. Acesso em 10 de junho de 2020.
[10] Dispositivo que converte, por exemplo, posição, velocidade, temperatura ou luz em um sinal elétrico.

Figura 1.4: Smartwatch e smartphone

Fonte: Repositório de imagens do *site* Freepick[11]

Uma outra aplicação bem rotineira na atualidade é o uso de *smartphones* e *smartwatches* (Figura 1.4). Esses são dotados, por exemplo, de acelerômetro[12], giroscópio[13], magnetômetro[14], sensor de luminosidade, barômetro[15] e sensor de proximidade. Em alguns sistemas, os dados fornecidos por esses sensores ajudam na geração de um número aleatório, proporcionando mais segurança e privacidade nesses dispositivos (HONG; LIU, 2015).

Os geradores de números aleatórios (RNGs) são a base de fortes medidas de segurança e privacidade. Com um número crescente de dispositivos inteligentes conectados à internet, a demanda por comunicação segura seguirá crescente.

Ao lançarmos um dado (não viesado e homogêneo) ao acaso, este determinará um número aleatório que é, de certa maneira, baseado nas leis da física clássica. Porém, existem outras forma de gerá-los que transcendem a física clássica.

Nessa direção, o *Contador de Geiger* (ou Geiger-Muller) é um instrumento utilizado para detectar e medir radiação ionizante, muito aplicado na dosimetria[16] de radiação, proteção radiológica, física experimental e indústria nuclear. Existem versões de contadores portáteis (e transportável com uma das mãos) e que detectam, por exemplo, partículas alfa, beta e raios gama utilizando o efeito de ionização produzido em um tubo

[11] Disponível em:< https://br.freepik.com>. Acesso em; 8 jul. 2020.
[12] Detecta o movimento baseado em eixos e mede a aceleração.
[13] Auxilia o acelerômetro na orientação em 360 graus.
[14] Mede campos magnéticos, utilizando a diferença de tensão.
[15] Mede a pressão do ar.
[16] Essencialmente é a determinação da dose de radiação em um ponto específico.

Geiger-Müller. Assim, a duração dos intervalos de tempo entre dois impulsos consecutivos do decaimento radioativo é imprevisível (ROHE, 2003). Essa imprevisibilidade é extremamente favorável à um TRNG.

A Distribuição de Chaves Quânticas ou *Quantum Key Distribution* (QKD) utiliza propriedades quânticas para trocar informações sigilosas, que podem ser utilizadas para criptografar dados e/ou mensagens. De maneira geral, a segurança do QKD é baseada no conhecimento de que o ato de medir um sistema quântico é o mesmo que perturbá-lo (HASAN *et al.*, 2017). Assim, caso a mensagem seja interceptada, uma troca quântica inevitavelmente deixará traços detectáveis, indicando a quebra do sigilo da mensagem.

O uso da Mecânica Quântica tem sido considerada como promissora na geração de algoritmos mais eficientes para a criptografia. Na pesquisa científica tem-se outros tipos de geradores, tais como a Ótica Quântica (MARANDI *et al.,* 2012) e o Vácuo Quântico (GABRIEL *et al.,* 2010).

Estas diferentes formas de se gerar números aleatórios nos parece distantes da nossa realidade, porém elas estão mais próximas e acessíveis do que imaginamos. Existem diversos *softwares* e páginas na internet que auxiliam o usuário a gerar números aleatórios verdadeiros de maneira gratuita e com determinada facilidade, sendo eles:

- **LavaRND**[17] - Criado em 2000, o LavaRND é um *software* livre que utiliza uma fonte caótica para alimentar a geração de números aleatórios de altíssima qualidade. Ele funciona em três etapas: (i) digitalização de uma fonte caótica através do LavaCan, na qual existe um chip CCD[18] de Webcam, que transforma uma imagem em 19.200 pixels, (ii) maximização da fonte de entropia, em que os pixels produzidos são principalmente de dados digitalizados que medem os níveis de ruídos de energia em um espaço completamente desprovido de luz e, a partir disso, (iii) a geração de números aleatórios por *software.*
- **Random.Org**[19] - Fundada em 1998 por Mads Haahr, a Random.org tem como objetivo disponibilizar gratuitamente, via internet, números aleatórios verdadeiros com base em ruídos atmosféricos. Pode-se gerar desde um número inteiro ou decimal até listas e sequências aleatórias. A Figura 1.5 mostra uma sequência composta por 100 números aleatórios verdadeiros, sem repetição, entre 1 e 100.

[17] www.lavarand.org
[18] É um sensor que captura a imagem em uma câmera.
[19] www.random.org

Figura 1.5: Sequência gerada no Random.Org

Fonte: o autor.

- **HotBits**[20] - Fundado em 1996, trata-se de um site que gera aleatórios verdadeiros por meio da incerteza das leis da mecânica quântica, em um processo de cronometrar pares sucessivos de decaimentos radioativos detectados por um tubo Geiger-Müller conectado a um computador. Basicamente, preenche-se um formulário de quantos *bytes* deseja gerar e o servidor retorna com os *bytes* aleatórios.

Números aleatórios imprevisíveis

Um Número Aleatório Imprevisível ou *Unpredictable Random Number Generator* (URNG) possui uma eficiência similar ao TRNG, todavia a fonte de entropia não é um fenômeno físico, mas sim alguma forma de interação entre o humano e a máquina. De acordo com Marton *et al.* (2012), os URNGs são uma aproximação prática dos TRNGs, pois os primeiros extraem a aleatoriedade de dispositivos de *hardware* periféricos (tais como sensores, mouses, teclados, etc) eletrônicos.

Apesar dos dispositivos periféricos serem, *a priori*, determinísticos, o comportamento humano aplicado a uma sequência determinística de operações impacta na complexidade da multiplicidade de eventos e parâmetros. Qualquer intervenção no processo de geração sequencial de dados perturbará o estado interno de um sistema de tal maneira que será praticamente impossível modelar e prever o resultado produzido. Em

[20] www.fourmilab.ch/hotbits/

suma, os URNGs baseiam-se na imprevisibilidade da interação humano-computador e no não-determinismo próprio da complexidade do fenômeno subjacente (MARTON *et al.*, 2012).

Um exemplo de aplicação do URNG está no *software Veracrypt*[21], que cria e mantém dispositivos de armazenamento em *Criptografia Imediata*[22] ou *On-the-fly Encryption*. Podem ser criptografados, mediante senha e chaves, arquivos, nomes de pastas, conteúdo de cada arquivo, espaço livre e metadados. O Veracrypt será instanciado mais adiante, após a abordagem sobre a Criptografia.

Números pseudo-aleatórios

> *Random numbers should not be generated with a method chosen at random* (Knuth, 2002).

Para entendermos melhor o que é um número pseudo-aleatório, vamos iniciar pela sequência $16838,908,17747,1817,18655,2726,19564,3634,20472,4543$ que denominaremos de *Sequência X*. Você saberia dizer se esta sequência é verdadeiramente aleatória (TRNG)? Se não for, o que significa não se ter uma aleatoriedade verdadeira? Para construirmos uma resposta, vamos construir alguns conceitos.

Um Gerador de Número Pseudo-aleatório ou *Pseudorandom Number Generator* (PRNG) é aquele que, em um primeiro momento, gera um número de forma arbitrária. Em um segundo momento, ao inspecionarmos uma sequência deles (e não apenas um único), concluímos que há uma *lei de formação* subjacente, que pode ser descoberta ou mesmo replicada. Em outras palavras, é um número (pseudo) aleatório gerado de forma determinística.

Para ilustrarmos o princípio de um PRNG, vamos construir um número na base binária que, subsequentemente possa ser transformado para uma base 10, formando números que podem ser representados na forma decimal (são exemplos: $1,2,56,78$) ou ponto-flutuante (são exemplos: $3.45 \times 10^2 = 0.345 \times 10^3 = 345$). Partindo dessas premissas, Chaitin (1990) traz uma situação-exemplo que ilustra muito bem a ideia da pseudo-aleatoriedade computacional (em inglês *Randomness*), consideremos duas sequências[23] numéricas:

$$\text{Sequência 1: } 01010101010101010101$$
$$\text{Sequência 2: } 01101100110111100010$$

Observando com cuidado, a Sequência 1 pode ser gerada por um algoritmo que imprime o número 01 dez vezes de forma sequencial e, computacionalmente, pode ser gerada por um *loop* (for) que vai de 1 a 10 pares. Como trata-se de um algoritmo, essa

[21] É um *software livre*, *fork* do *TrueCrypt* (seu antecessor). Vem sendo desenvolvido desde 2013 e pode ser obtido gratuitamente em www.veracrypt.fr.

[22] Ou Criptografia em Tempo Real é aquela em que os dados são encriptografados imediatamente antes de serem salvos e descriptografados logo após serem carregados, tudo de forma automática, sem intervenção direta do usuário.

[23] Estas podem ser transformadas em dois números na base 10. Nesse caso, $(01010101010101010101)_2 = (349525)_{10}$ e $(01101100110111100010)_2 = (445922)_{10}$.

sequência poderia ter qualquer comprimento finito (limitado somente pela precisão numérica da linguagem de programação), por exemplo, 1 milhão de pares (se há o aumento do comprimento da sequência, então espera-se um maior tempo de processamento).

Por outro lado, a Sequência 2 nos parece que não possui uma regra ou lei de formação, nos indicando simplesmente o armazenamento dessa sequência em si. Portanto, se a sequência fosse expandida para 1 milhão de termos, o algoritmo teria, pelo menos, o comprimento dessa sequência.

Para Chaitin (1990) essa *incompressibilidade* da Sequência 2 é uma propriedade dos números aleatórios. Aleatoriedade é definida [24] em termos da incompressibilidade (CHAITIN, 1990) da seguinte forma: "Uma sequência de números é aleatória se o menor algoritmo tem, aproximadamente, o mesmo número de bits de informação da própria sequência" (CHAITIN, 1990, p. 14).

Para o referido autor, essa definição algorítmica de aleatoriedade, em alguma instância, substitui os conceitos clássicos da teoria da probabilidade que, sucintamente, é baseada em um conjunto de possibilidades do qual é atribuída uma probabilidade.

O Código 1.3 traz a implementação do comando rand, que é um dos algoritmos mais simples para se gerar um PRNG inteiro[25] entre 0 e 32767.

Código 1.3: Algoritmo rand em linguagem C.

```
unsigned long int next = 1;
int rand(void)
        {
        next = next * 1103515245 + 12345;
        return (unsigned int)(next/65536) % 32768;
        }
```

Fonte: adaptado de (Kernighan e Ritchie, 1988, p. 46).

Buscando compreender como o número é gerado, iremos transformá-lo em pseudocódigo. O resultado da tração para o Portugol consta no Código 1.4.

Código 1.4: Pseudocódigo que gera um pseudo-aleatório.

```
Pseudocódigo GeraAleatorioInteiro
inicio
        proximo = 1
        proximo = proximo * 1103515245 + 12345
        aleatorio = truncar(proximo/65536)
        aleatorio = aleatorio % 32768
        escreva("O aleatório é",aleatorio)
fim
```

[24] Foi proposta, independentemente, por volta de 1965 por A. N. Kolmogorov da Academia de Ciências da URSS e G. J. Chaitin quando estava na graduação no City College da City University de Nova York. Ambos desconheciam a proposta de Ray J. Solomonoff, em 1960, em um esforço para medir a simplicidade das teorias científicas (CHAITIN, 1990).

[25] Com efeito esclarecedor, em C uma variável do tipo int pode assumir valores entre -32768 a 32767 e, por isso, o referido algoritmo tem esta limitação. Tal limitação pode ser expandida se utilizarmos o tipo long, que vai de -9223372036854775808 a 9223372036854775807. O tipo unsigned int é usado para inteiros positivos e vai de 0 a 65535.

Um algoritmo TRNG necessita de uma condição inicial ou semente (*seed*). Observando o Código 1.4, atribui-se à variável proximo o valor 1, que é de inicialização. Em seguida, atualiza-se a referida variável com o valor $1 \times 1103515245 + 12345 = 110352759$. Na próxima linha, $1103527590 \div 65536 = 16838.494720459$. A atualização da variável aleatorio é dada pela parte inteira, ou seja, 16838. Finalmente, o símbolo $\%$ é a operação módulo[26], portanto, $16838 \ \% \ 32768 = 16838$.

Observe atentamente que se a semente próximo = 1 não for alterada, o algoritmo gerará sempre o número 16838. Podemos comparar, matematicamente, com uma função: dado o domínio (semente), a imagem é um número pseudo-aleatório. Claro que o Código 1.4 é bastante simples, mas existem outros que são mais complexos e aumentam muito a variabilidade da aleatoriedade. A conclusão é que a semente determina a aleatoriedade e a função matemática o seu nível.

Do ponto de vista formal, L'Ecuyer (1990) define um gerador pseudo-aleatório como um estado que envolve um espaço finito $\mathbb{S}$, μ uma distribuição inicial de probabilidade em $\mathbb{S}$, a *função de transição* $f: \mathbb{S} \rightarrow \mathbb{S}$, um conjunto finito de símbolos $\mathbb{U}$ e uma função de saída $g: \mathbb{S} \rightarrow \mathbb{U}$. A partir disso, inicia-se a geração tomando uma semente $s_0 \in \mathbb{S}$ de acordo com μ e aplicando $\mu_0 = g(s_0)$. Em seguida itera-se, de $i = 1, \dots, s_i = f(s_{i-1})$ e $\mu_i = g(s_i)$. Para aprofundamento de aspectos formais de um TRNG, recomendamos a leitura de L'Ecuyer (1999).

Mas afinal, qual é o impacto da semente em um PRGN? Para que exista uma boa aleatoriedade, é fundamental que o algoritmo seja executado sempre com diferentes sementes, como propõe o Código 1.5.

Código 1.5: Implementação da semente em linguagem C.

```
void srand2(unsigned int seed) {
        next = seed;
        }
```

Para aumentar a aleatoriedade, troca-se a primeira linha do Código 1.3 (next = 1) pelo Código 1.5. Por exemplo, se seed = 1, então o resultado será 16838. Quando mudamos para seed = 2, o resultado será 908.

Retomando a *Sequência X*, presente no inicio desta seção, observe que ela foi gerada a partir da execução do Código 1.3 com seed = 1. Portanto, afirmamos que não é TRNG, dado que existe uma "lei de formação" e a semente é fixada.

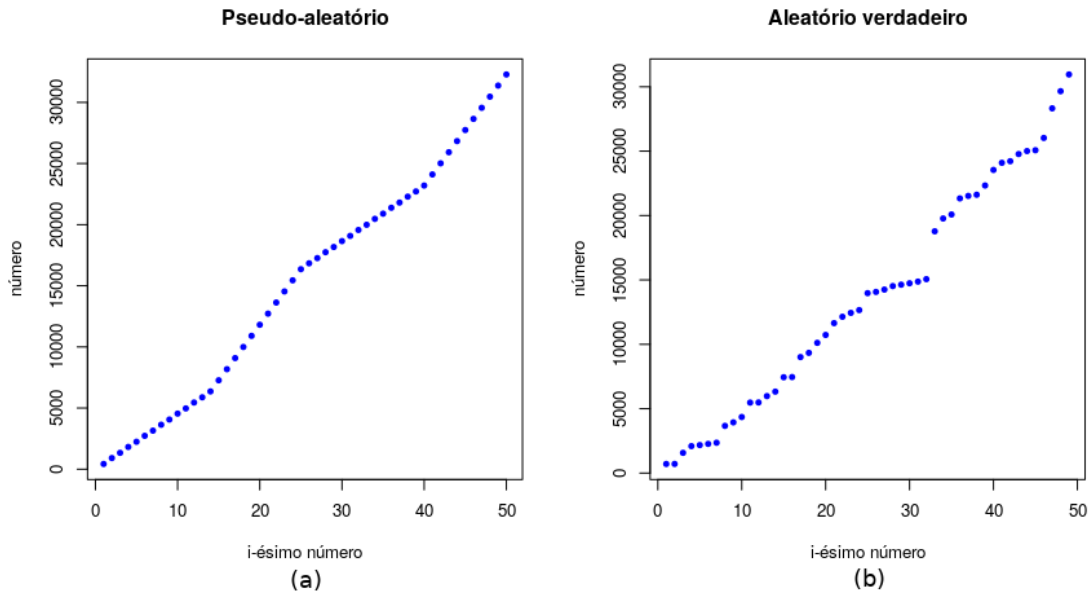
Apesar da semente nos parecer um complicador para aleatoriedade, existem algumas formas de aplicar determinada variabilidade, gerando sementes de formas não sequenciais. Uma delas é utilizar alguma informação do horário ou data (*time/date*) do computador ou uma composição delas. No caso da linguagem C, a função time retorna o tempo decorrido, em segundos, desde as 00:00:00 (hora/minuto/segundo) no UTC[27] de 1^o de janeiro de 1970. Esta contagem de tempo é denominada *Unix Timestamp* e permite representar uma data e hora em um número inteiro. Por exemplo, o dia 01/01/2020 as

[26] Retorna o resto da divisão entre dois números inteiros. Por exemplo, 7 modulo 3 = 1.
[27] O Coordinated Universal Time é um horário fixo mundial, considerado o sucessor do Greenwich Mean Time (GMT).

12h00 corresponde ao 1577880000. Já a véspera de Natal (24/12, 23h59) do ano de 1981 corresponde ao 378086399.

A literatura apresenta muitas discussões e resultados sobre TRNGs e PRNGs e como eles podem ser diferenciados a partir de sequências numéricas. Para isso, elaboramos algumas simulações numéricas, com a finalidade de comparar sequências compostas por aleatórios verdadeiros e pseudos. Nessa direção, a Figura 1.6 mostra duas sequências numéricas (ordenadas), compostas cada uma por 50 números inteiros aleatórios entre 1 e 32767. O Gráfico (a) é o resultado do Código 1.3 (pseudo-aleatório), verificando que a sequência gerada segue um padrão, diferentemente do Gráfico (b), que vem de uma fonte verdadeiramente aleatória.

Figura 1.6: Gráficos dos 100-ésimos termos da sequência de aleatórios.

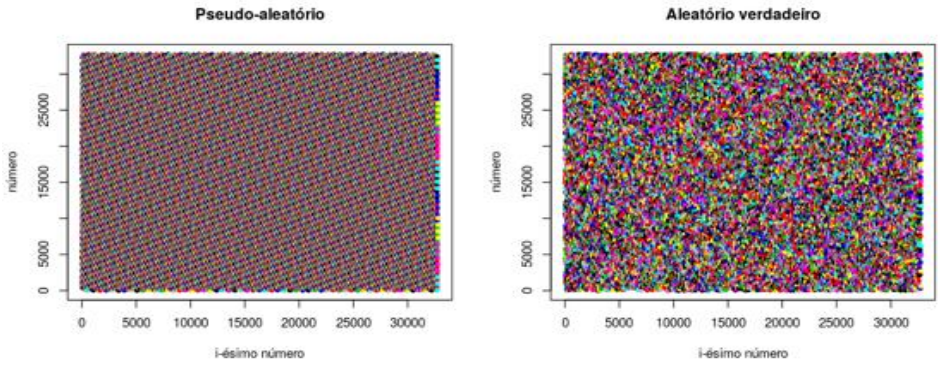


Fonte: o autor.

Para enfatizar ainda mais a diferença entre os TRNGs e PRNGs, realizamos uma segunda simulação. A Figura 1.7 mostra duas sequências numéricas (ordenadas), compostas cada uma por 32767 números inteiros aleatórios entre 1 e 32767, utilizando a técnica de blocos de colorização (10 cores em blocos de 10). Essa técnica consiste em um gráfico de dispersão em que cada dado (x,y) é colorido, distribuídos em um determinado intervalo de cores.

O Gráfico (a) é o resultado do Código 1 e o Gráfico (b) é o resultado gerado no Random.Org. Notoriamente demonstra-se a diferença entre uma sequência de pseudo-aleatórios e aleatórios verdadeiros.

Figura 1.7: Gráficos dos 32767-ésimos termos da sequência de aleatórios



Fonte: o autor.

Números quasi-aleatórios

Números aleatórios podem ser determinados a partir de modelos especificados e que satisfazem um conjunto de testes aceito, simulando um determinado nível de aleatoriedade, como é o caso dos pseudo-aleatórios. Aqueles gerados a partir de sequências Quasi-Aleatórias (GENTLE, 2004) buscam, fundamentalmente, uma distribuição o mais uniforme possível dos números gerados, sendo denominados de *Quasirandom Number Generator* (QNG).

De acordo com Soboí (1990), números que são utilizados em alguns algoritmos de Monte Carlo para obter convergência são quasi-aleatórios. Os métodos de Monte Carlo consistem em gerar aleatórios para distribuições de probabilidades, dentro de um modelo, com o objetivo de "produzir centenas ou milhares de cenários" (SOUZA, 2006). A eficiência desses métodos pode ser melhorada aumentando o número de iterações. As simulações de Monte Carlo são amplamente utilizadas na teoria financeira, com destaque ao cálculo das opções, na medição de risco de mercado e de crédito, no cálculo do Valor em Risco (VaR), na análise de projetos de investimento e na solução das Opções Reais (SOUZA, 2006).

A literatura apresenta diversos algoritmos de QNGs, e como o objetivo desse texto não é aprofundarmos em cada um deles (uma vez que não é objetivo desse texto trazer a estrutura estatística e matemática rigorosa), deixaremos as referências para posterior consulta pelo leitor. Os algoritmos são: *Linear Congruential Generators* (PARK; MILLER, 1988), *Multiple Recursive Generators* (KNUTH, 2002), *Mersenne-Twister* (MATSUMOTO; NISHIMURA, 1998) e o *Well Equidistributed Long-period Linear Generators* (PANNETON; L'ECUYER; MATSUMOTO, 2006).

Aplicações

As aplicações dos números aleatórios gerados por computador são as mais variadas (algumas delas já foram mencionadas ao longo desse texto). Em Ciências da Computação os aleatórios são amplamente utilizados em simulações gráficas, aprendizado de máquina,

inteligência artificial (IA), jogos eletrônicos (incluindo os consoles dedicados a jogos) e Criptografia.

Na Estatística, são utilizados, por exemplo, em simulações numéricas probabilísticas, experimentação com amostras geradas aleatoriamente, estimativas para os métodos de Monte Carlo, combinatória e *shuffling* (embaralhamento) em jogos de cartas em casinos.

Um exemplo pontual nas áreas de Bioquímica e Biologia Molecular é o *DNA-based circuit design*, na qual as tecnologias tradicionais baseadas em silício são substituídas por fenômenos naturais (GEARHEART; ARAZI; ROUCHKA, 2010) com o objetivo de se construir sequências aleatórias de DNA.

Especificamente para as áreas que envolvam seres vivos (são exemplos: sociologia, psicologia, biologia, medicina, farmácia-bioquímica e educação), a seleção dos agentes (denominados em cada área como participantes, sujeitos, cobaias, pacientes, indivíduos, estudantes, etc) pode ser gerada aleatoriamente, combinando aspectos qualitativos e quantitativos reunidos em um algoritmo, por exemplo. Nesse cenário é importante produzir uma amostra representativa que reflita as características da população, inferindo para uma possível generalização.

Uma importante aplicação dos PRNGs estão na área da Avaliação em Larga Escala, especificamente na *Teoria de Resposta ao Item* (TRI). Para Piton-Gonçalves e Aluísio (2015) a TRI propõe uma modelagem estatístico-matemática que busca relacionar a probabilidade de um examinado responder corretamente a um item, dada sua habilidade em um teste educacional (múltipla escolha, por exemplo). Dentre as etapas de validação do instrumento (teste educacional), realiza-se experimentos computacionais que simulam[28] as respostas de cada examinado em cada item (questão do teste). Tais respostas são obtidas a partir de PRNGs e QNGs, que devem seguir determinadas métricas previamente estabelecidas. Uma aplicação da TRI está no Exame Nacional do Ensino Médio (ENEM).

Tradicionalmente, as máquinas dotadas de IA trabalham com os raciocínios dedutivo e indutivo. Por outro lado, enquanto tendência mundial, Takefuji (2018) aponta que o *Deep Learning*[29] é classificado em raciocínio indutivo e estocástico[30], que se baseia em números aleatórios. Para o referido autor, antes de executar uma IA por Deep Learning, a semente do número aleatório deve ser corrigida, uma vez que os problemas de reprodutibilidade de ações/interpretações realizadas pela IA podem ser corrigidas.

A técnica de geração de dados (aleatórios) simulados é muito utilizada quando queremos validar ou simular variáveis em ciências exatas, humanas ou biológicas. Em suma, são inúmeras as aplicações dos números aleatórios decorrentes de procedimentos e processos computacionais. Visando instanciar uma das aplicações mais importantes e rotineiras da aleatoriedade em Ciências da Computação, dedicamos a próxima seção para a Criptografia.

[28] Na estatística é a técnica de Geração de Dados para validação ou simulação.
[29] É uma área da Aprendizagem de Máquina que foca em algoritmos inspirados nas estruturas e funcionamentos do cérebro humano.
[30] Que depende do acaso.

Criptografia e números aleatórios

É de longa data que o homem necessita da privacidade, que vai desde a proteção do seu próprio corpo e lar até as suas ideias do ponto de vista intelectual. De acordo com Holvast (2009), os advogados americanos Samuel Warren e Louis Brandeis descreveram o direito à privacidade com a frase celebre "*the right to be let alone"* (WARREN; BRANDEIS, 1890). Em 1967, a publicação do *Alan Westin's Privacy and Freedom* definiu a privacidade em termos de autodeterminação: "privacidade é a reivindicação de indivíduos, grupos ou instituições para determinar, por si mesmos, quando como e em que medida as informações sobre eles é comunicado aos outros".

Com o avanço das telecomunicações, o envio de mensagens textuais necessitava de mecanismos para ocultá-las, mantendo ao máximo a privacidade entre o remetente e o destinatário. Com isso, surgem os *códigos* e as *cifras* que são elementos importantíssimos para a Criptografia[31]. O Código é o ato de substituir frases ou números por outros de forma direta. Por exemplo, todas as vezes que for dita a frase "tudo azul" significa que "estou indo escovar os dentes". Em um processo muito mais complexo, a cifra transforma os caracteres de uma mensagem em outra ilegível.

Historicamente, a Cifra de César é de substituição e seu princípio é deslocar o alfabeto em um determinado número de posições. Um exemplo é a cifra com treze rotações, denominada de ROT13[32] que faz a correspondência $A \leftrightarrow N$, $B \leftrightarrow O$ e assim por diante. Ou seja, a cadeia de caracteres ABCDEFGHIJKLMNOPQRSTUVWXYZ é cifrada como NOPQRSTUVWXYZABCDEFGHIJKLM. Tal cifragem é a *chave criptográfica* ou simplesmente *chave* que *encripta* (ou cifra) uma mensagem. Por exemplo, vamos encriptar a frase citada de Von Neumann (no ínicio desse texto):

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

Aplicando o ROT13, temos a seguinte mensagem ao destinatário:

Nalbar jub pbafvqref nevguzrgvpny zrgubqf bs cebqhpvat enaqbz qvtvgf vf, bs pbhefr, va n fgngr bs fva.

Uma vez que o destinatário conheça a chave, torna-se viável a *decriptação* da mensagem. Ao inverter a chave e recupera-se a mensagem original. Os passos que foram seguidos para encriptá-la são denominados de Algoritmo de Criptografia.

Vimos anteriormente que a maior parte dos geradores de números aleatórios são utilizados em criptografia. Portanto, afirmamos que todos os sistemas digitais que utilizam algum nível de comunicação (direto ou indireto) necessitam de criptografia para manter o sigilo ou a privacidade. São múltiplos exemplos: ao acessar uma rede *wi-fi*, ao utilizar um dispositivo *bluetooth*, ao acessar um aplicativo de *smartphone*, ao utilizar seu veículo. Poderíamos citar páginas e páginas de exemplos e ainda assim não seriam suficientes para a quantidade de aplicações.

[31] Do grego *kriptós* (oculto) e *grápho* (grafia), tem como objetivo transmitir sigilosamente uma informação.
[32] O site https://rot13.com disponibiliza a cifra para testes.

A pergunta que você pode estar fazendo nesse momento é "Qual é a relação entre a criptografia e aleatoriedade?". Para respondê-la, precisamos compreender o conceito do nível de *segurança* na transmissão de dados e informações.

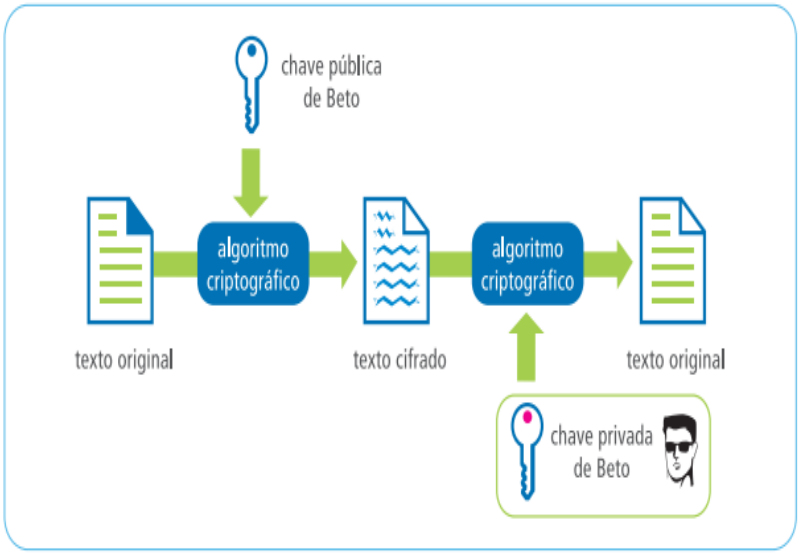
Um princípio da criptografia é que, em caso de interceptação, a mensagem não seja revelada ou decodificada. Porém, na prática, existem pessoas que se dedicam a quebra de chaves criptográficas em sistemas de segurança ao redor do mundo. São os chamados *Crackers*, que buscam vulnerabilidades nos algoritmos de criptografia, com a finalidade de decodificarem as mensagens e até mesmo descobrirem a chave do algoritmo. Visando coibir ações como essa, cada vez buscam-se algoritmos mais eficientes, garantindo uma transmissão mais segura de dados. Por exemplo, jamais poderíamos utilizar o ROT13 na prática, posto que com 25 tentativas (no máximo) a chave é decodificada.

Somada à possibilidade de interceptação, tem-se o problema de transmissão da chave pelo remetente ao destinatário, que não pode ser explícita. Nos parece um paradoxo, pois mesmo que a chave codifique a mensagem, quem irá codificar a chave? Pensando nisso, Rivest, Shamir e Adleman (1978) publicaram um dos algoritmos mais importantes e utilizados na atualidade que levam as iniciais dos seus criadores: o RSA[33].

O RSA é o primeiro sistema de criptografia que permite que os remetentes assinem suas mensagens enviadas, de forma que o destinatário reconheça tal assinatura. É classificado como um algoritmo de *Criptografia Assimétrica* composto por uma *Chave Pública* e uma *Chave Privada*. A primeira é de conhecimento público, podendo ser amplamente divulgada (inclusive em páginas públicas na internet) e está associada ao remetente. A segunda é de conhecimento privado, devendo ser mantida em sigilo.

Para cifrar uma mensagem no RSA basta utilizar a chave pública do destinatário que decifrará com sua chave privada, conforme ilustra a Figura 1.8. Inicialmente nos parece um algoritmo inviável, porém veremos que o seu sucesso se deve às chaves que dependem de um gerador de números aleatórios eficiente.

Figura 1.8: Esquema de funcionamento do RSA.



Fonte: Ladeira e Raugust (2017)

[33] Rivest, Shamir e Adleman.

Inicialmente o algoritmo deve determinar dois números primos grandes. Para isso, o método mais simples conhecido é a *Divisão por Tentativa*, em que dado um número natural n, busca-se algum divisor diferente dele próprio. Por exemplo, o 17 tem somente os divisores 1 e 17 e, por isso, é primo. Isso não ocorre com 10, que possui os divisores 2 e 5, além do 1 e 10. Em um primeiro momento, nos parece um excelente método para gerar dois primos, porém, quando tratamos de primos grandes, o tempo computacional para a ser extremamente relevante. Por exemplo, para determinar se o número $266058405243152137503389944709$ (com 30 dígitos) é primo ou composto[34], necessitaríamos testá-lo $266058405243152137503389944708$ vezes, sendo extremamente lento em situações práticas, principalmente no envio e recebimento de dados constantes ao longo da internet. Claro que poderíamos otimizar o algoritmo, reduzindo o espaço de busca somente para os ímpares, todavia o tempo computacional ainda seria demasiado.

Buscando métodos mais eficazes e rápidos, a área de *Testes de Primalidade* vem trazendo diversas formas de se decidir se um número natural n é primo. Citamos alguns deles: *Primalidade de Fermat*, *Solovay-Strassen* (SOLOVAY; STRASSEN, 1977), *Crivo de Eratóstenes* (HORSLEY, 1772), *Crivo de Atkin*[35] (ATKIN; BERNSTEIN, 2004) e o *Elliptic Curve Primality Proving*[36] (ADLEMAN; HUANG, 1987). Outros testes são citados e estudados, tais como (DUTA; GHEORGHE; TAPUS, 2015): *Agrawal-Kayal-Saxena* (AKS), *Lucas-Lehmer*, *Baillie-PWS*, *Pepin's test*, *Lucas-Lehmer-Riesel* (LLR), *Proth's theorem*, *Quadratic Frobenius*, *Adleman-Pomerance-Rumley* (APR), *Lucas* e *Pocklington*.

Miller (1976) apresentou dois algoritmos determinísticos que testam a primalidade de um número assumindo a Hipótese de Riemann[37] estendida e a função φ de Euler, que é computacionalmente equivalente a fatorar números inteiros. Quatro anos mais tarde, Rabin (1980) apresentou um algoritmo probabilístico[38] com alto grau de acerto, que é baseado em Miller. Essa junção culminou no conhecido *Teste de Primalidade de Miller-Rabin* (LADEIRA; RAUGUST, 2017), que se utiliza dos princípios da aleatoriedade para testar se um número é primo ou composto. Se um número é composto, a probabilidade de retornar falso é de $\frac{1}{4}$ para cada teste. Se repetirmos 30 testes, então a probabilidade de falhar será $\frac{1}{4^{30}}$ que é, aproximadamente, 8.67×10^{-19}. Um baixo número de testes aliado a uma baixa probabilidade de erro fazem com que Miller-Rabin seja eficiente em termos de aplicabilidade em criptografia. Essencialmente, um gerador de números primos aleatórios deve testar os números (grandes, de 512 a 1024 bits) quanto a primalidade.

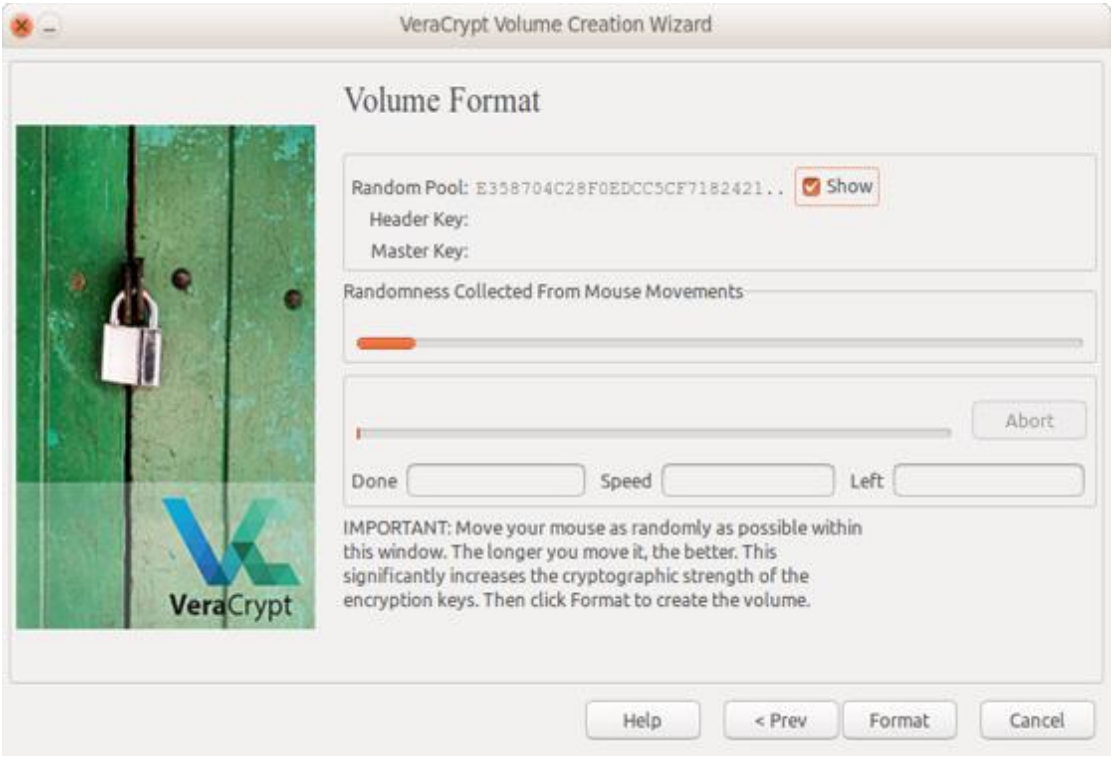
[34] Aquele que não é primo. Todos pares maiores do que o 2 são compostos, por exemplo. 2 é o único primo par.
[35] É um algoritmo derivado do Crivo de Erastóstenes, com melhor comportamento assintótico.
[36] O teste mostrou que o número $(((((((((2^3+3)^3+30)^3+6)^3+80)^3+12)^3+450)^3+894)^3+3636)^3+70756)^3+97220$ é primo e possui 20.562 dígitos. Para prová-lo, foram utilizados um conjunto de computadores em computação distribuída; e o cálculo iniciou em setembro de 2005 e finalizou em junho de 2006. O tempo de CPU acumulado é o equivalente a 2.39 GHz por 2219 dias, conforme aponta o site *The Largest Know Primes* - https://primes.utm.edu/primes.
[37] Relaciona a função zeta aos números primos.
[38] Pertencente à classe de algoritmos de Monte Carlo.

Voltando ao *software Veracrypt*, este possui um gerador de números aleatórios que gera a chave de criptografia primária, a chave secundária (modo XTS[39]), o *salt*[40] e arquivos auxiliares da chave. Ele cria um conjunto de valores aleatórios na memória RAM com 320 bytes de comprimento e é gerado a partir de uma combinação de números gerados por:

- **PRNGs**. Os valores podem ser gerados pelo sistema operacional, estatísticas do tráfego da rede de internet e contadores de tempo do sistema (data, hora, tempo que o computador está ligado, etc).
- **URNGs**. Movimentos do mouse e teclado.

Com foco na aplicação da geração do número aleatório, a Figura 1.9 mostra a tela que cria o *Volume* (disco a ser encriptado) e será formatado aplicando criptografia. Nesse momento o usuário simplesmente deve movimentar o mouse ao acaso (*randomness collected from mouse movements*) pelo tempo que o usuário achar necessário (quanto maior o tempo de movimentação, maior a aleatoriedade) alimentando a sequência *random pool* que gerará as chaves *header* e *master* utilizadas na encriptação.

Figura 1.9: Tela do Veracrypt para a geração do número aleatório.

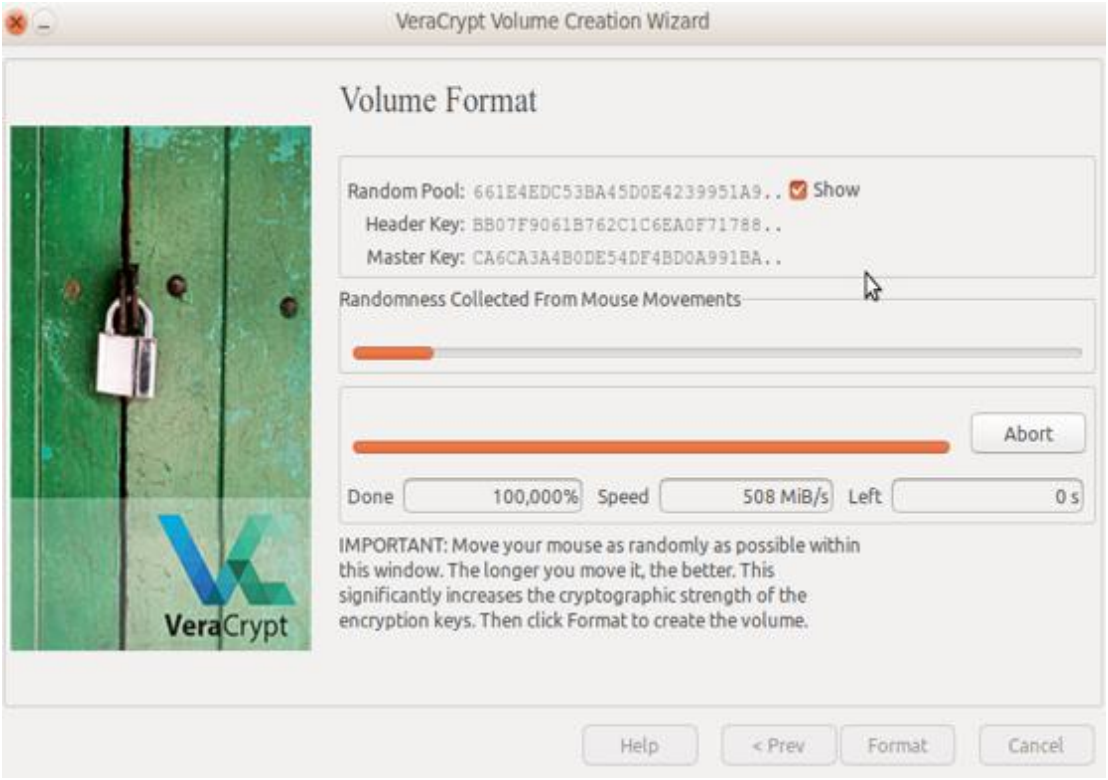


Fonte: o autor.

Uma vez que o processo de geração for finalizado, as chaves serão parcialmente mostradas, indicando sucesso no procedimento, conforme mostra a Figura 1.10.

[39] O *XEX-based tweaked-codebook mode with ciphertext stealing* é um modo de operação de cifragem em bloco para criptografia de um disco completo, por exemplo um disco rígido ou um pendrive.
[40] O "sal" é uma função matemática de sentido único, que evita que duas senhas idênticas produzam embaralhamentos idênticos.

Figura 1.10: Tela do Veracrypt com as chaves geradas.



Fonte: o autor.

O sistema de encriptação de dados Veracrypt é mundialmente utilizado e, além de criptografar o disco utilizando URNGs, ainda permite o uso de cinco diferentes algoritmos criptográficos e dez combinações em cascata[41]. É um *software* que pode ser utilizado por qualquer nível de usuário, em diferentes sistemas operacionais e pode ser lido por *smartphones*. Com ele é possível, inclusive, criptografar uma unidade de memória sólida (pendrive, cartão de memória ou unidade de estado sólido - SSD), aumentando a segurança no caso de perda ou extravio.

Curiosidades

Existiram (e provavelmente ainda existam) situações em que ou o gerador de aleatórios foi mal escolhido, ou mal aplicado ou inadequado. Trazemos aqui algumas curiosidades sobre o uso e aplicação destes números em situações bastante curiosas e que causaram grandes prejuízos a governos.

- ***Canada's random immigration lottery***.[42] Em 2017 o Canadá introduziu um novo sistema que estendia o *status* de residente permanente aos pais e avós aos cidadãos canadenses. O processo foi projetado para selecionar candidatos aleatoriamente, a fim de tornar o processo mais justo do que um sistema anterior. Para isso, utilizaram o *Microsoft Excel* (que gera um PRNG e não um TRNG) para

[41] Ou seja, criptografia sob criptografia, o que diminui, em muito, a possibilidade de quebra.
[42] gizmodo.com/canadas-random-immigration-lottery-uses-microsoft-excel-1826711895

sortear 10000 pessoas com *status* de residente permanente em uma população de cerca de 100000. Não se sabe qual foi a versão[43] do Excel utilizada na loteria, mas o estudo de McCullough (2008) apontou que o Excel 2007 era defasado e não passaria nos testes de aleatoriedade. Uma possibilidade é de que tenham utilizado uma versão mais antiga do Excel.

- ***Hot Lotto fraud scandal***.[44] [45] Considerado um dos maiores golpes em loterias americanas, Eddie Raymond Tipton - ex-diretor da *Multi-State Lottery Association* - manipulou os computadores que geravam aleatórios, garantindo a ele milhões de dólares em pagamentos a partir de jogos fraudulentos. Segundo o processo jurídico, Tipton tinha acesso aos computadores e instalou um *root kit*[46]. Segundo os registros da época, a fraude iniciou em 13 de abril de 2015 e finalizou em 20 julho de 2015. Recentemente foi condenado a 25 anos de prisão em Iowa e confessou manipular sorteios nos estados de Iowa, Colorado, Wisconsin, Kansas e Oklahoma.
- ***Plague Arizona lottery***.[47] Em meados de agosto de 2013, os computadores que sorteavam os números dos bilhetes da loteria *Pick 3* (semelhante a Loto) falharam no algoritmo que gerou os números aleatórios no sorteio daquele mês, de forma que os números 8 e 9 não eram sorteados em certas combinações numéricas. Com isso, 92.3% dos bilhetes tinham chance em vencer e 7.7% em não vencer. Não sendo o único caso, em 2017, computadores que foram utilizados no mesmo estado, geraram em um sorteio os números 1-4-8-12-28 e na semana seguinte, os mesmos foram sorteados. Até onde se sabe, tais computadores deixaram de ser utilizados com alegação de "falhas técnicas" e os sorteios subsequentes seguiram normalmente. Críticos (CLAYWORTH, 2019) alegam que ocorrem diversos sorteios idênticos ao longo dos EUA, identificando problemas com geradores de aleatórios, causando um prejuízo de cerca de US$ 80.5 bilhões anuais.

Referências

ADLEMAN, L.; HUANG, M.-D. Recognizing primes in random polynomial time. In: **[S.l.: s.n.]**, 1987. p. 462-469.

ATKIN, A.; BERNSTEIN, D. Prime sieves using binary quadratic forms. **Math. Comput.**, v. 73, n. 246, p. 1023–1030, Rhode Island, abr. 2004.

CAMARA, C. et al. Design and analysis of a true random number generator based on

[43] Desde a versão do Excel 2010, ele se utiliza do *Mersenne Twister Algorithm* (MT19937) para gerar aleatórios, que é considerado eficiente.
[44] claimchoice.blogspot.com/2018/06/hot-lotto-fraud-scandal_13.html
[45] privacysecuritybrainiacs.com/privacy-professor-blog/eddie-tipton/
[46] É um *software* de rápida instalação que altera um sistema e depois de algum tempo ele pode se autodestruir sem deixar rastros.
[47] arstechnica.com/tech-policy/2017/10/1-4-8-12-28-glitch-causes-same-lottery-numbers-to-hit-multiple-times-in-arizona /

gsr signals for body sensor networks. **Sensors**, v. 19, n. 9, p. 2033, Switzerland, abr. 2019. ISSN 1424-8220.

CHAITIN, G. J. **Information, Randomness and Incompleteness:** papers on Algorithmic Information Theory, 2nd Edition. [S.l.]: World Scientific, 1990.

CLAYWORTH, J. Identical winning numbers crop up in hundreds of U.S. lotteries. Are the drawings really random? The Des Moines Register, jan. 31, 2019. Disponível em: <https://www.desmoinesregister.com/story/news/investigations/2019/01/31/united-states-lottery-numbers-broken-system-identical-drawings-controversy/1863102002/>. Acesso em: 17. fev de 2020.

COHN, H. **Advanced Number Theory**. [S.l.]: Dover Publications, 1980. (Dover Books on Mathematics Series).

CORMEN, T.; LEISERSON, C.; STEIN, R. **Algoritmos**: teoria e prática. [S.l.]: Elsevier Editora, 2012.

DAVIES, R. Hardware random number generators. 15th Australian Statistics Conference. **51st conference of the NZ Statistical Association**, v. 15, n. 1, p. 1-10, New Zealand, set. 2000.

DUTA, C.; GHEORGHE, L.; TAPUS, N. Framework for evaluation and comparison of primality testing algorithms. In: 20TH INTERNATIONAL CONFERENCE ON CONTROL SYSTEMS AND COMPUTER SCIENCE, 2015, . **Anais** ... United States, 2015. p. 483-490.

DUTANG, C.; WÜRTZ, D. A note on random number generation. 2009. Disponível em: <https://cran.r-project.org/web/packages/randtoolbox/vignettes/fullpres.pdf>. Acesso em: 12 de out. de 2019.

GABRIEL, C. et al. A generator for unique quantum random numbers based on vacuum states. **Nature Photonics, Nature Publishing Group**, v. 4, n. 10, p. 711-715, Reino Unido, 2010.

GEARHEART, C. M.; ARAZI, B.; ROUCHKA, E. C. Dna-based random number generation in security circuitry. **Biosystems**, v. 100, n. 3, p. 208-214, Rockville Pike Bethesda, mar. 2010.

GENTLE, J. E. **Random Number Generation and Monte Carlo Methods**. 2. ed. [S.l.]: Springer, 2004.

HASAN, R. et al. A true random number generator based on the photon arrival time registered in a coincidence window between two single-photon counting modules. **Chinese Journal of Physics**, v. 56, n. 1, p. 385-391, Taiwan, nov. 2017.

HOLVAST, J. History of privacy. In: Matyáš V., Fischer-Hübner S., Cvrček D., Švenda P. (Eds). **The Future of Identity in the Information Society 2008**. Heidelberg: Springer Berlin Heidelberg, v. 29, [s.n.], p. 13-42, 2009.

HONG, S. L.; LIU, C. Sensor-based random number generator seeding. **IEEE Access**, **[S.l.: s.n.]**, v. 3, p. 562-568, mai. 2015.

HORSLEY, S. Koekinon epatos0enot2; or, the sieve of eratosthenes. being an account of his method of finding all the prime numbers, by the rev. samuel horsley, f. r. s. Philosophical Transactions (1683-1775). **The Royal Society**, v. 62, p. 327-347, 1772.

HORSLEY, S. Κοσκινον Ερατοσθενους or, The Sieve of Eratosthenes. Being an Account of His Method of Finding All the Prime Numbers, by the Rev. Samuel Horsley, F. R. S., **Philosophical Transactions**, v. 62., [s.n] , p. 327-347, Londres, 1772.

KERNIGHAN, B. W.; RITCHIE, D. M. **The C Programming Language**. 2. ed. [S.l.]: Prentice Hall Professional Technical Reference, 1988. ISBN 0131103709.

KNUTH, D. E. **Art of Computer Programming, Volume 2: Seminumerical Algorithms**. 3. ed. USA: Addison Wesley Longman Publishing Co., Inc., 2002.

LADEIRA, R.; RAUGUST, A. Uma análise da complexidade do algoritmo RSA implementado com o teste probabilístico de Miller-Rabin. **Revista de Empreendedorismo, Inovação e Tecnologia**, v. 4, n. 1, p. 24-33, Passo Fundo, jan.-jun, 2017.

L'ECUYER, P. Random numbers for simulation. **Communications of the ACM**. v. 33, n. 10, p. 85-97, New York, ago. 1990.

_____. Uniform random number generators. **Proceedings of the 1997 Winter Simulation Conference**. Ed. S. Andradóttir, K. J. Healy, D. H. Withers, and B. L. Nelson. [s.n], p. 127-133, Atlanta, Georgia, dec. 1997.

MACHALE, D. **Comic sections:** The book of mathematical jokes, humour, wit, and wisdom. Boole Press, 1st edition, 154 p. 1993.

MARANDI, A. et al. All-optical quantum random bit generation from intrinsically binary phase of parametric oscillators. **Optics Express**, v. 20, n. 17, p. 1-9, USA, jun. 2012.

MARTON, K. et al. Generation and testing of random numbers for cryptographic applications. **Proceedings of the Romanian Academy - Series A**: Mathematics, Physics, Technical Sciences, Information Science, v. 13, n. 4, p. 368-377, Bucharest, Romania, ago. 2012.

MATSUMOTO, M.; NISHIMURA, T. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. ACM Trans. Model. Comput. Simul. **Association for Computing Machinery**, New York, v. 8, n. 1, p. 3-30, jan. 1998.

MCCULLOUGH, B.D. Microsoft Excel's 'Not The Wichmann–Hill' random number generators. **Computational Statistics & Data Analysis**, v. 52, n. 10, p. 4587-4593, USA, jun. 2008.

MILLER, G. L. Riemann's hypothesis and tests for primality. **Journal of Computer and System Sciences**, v. 13, n. 3, p. 300-317, 1976.

PANNETON, F.; L'ECUYER, P.; MATSUMOTO, M. Improved long-period generators based on linear recurrences modulo 2. ACM Trans. Math. Softw. **Association for Computing Machinery**, New York, v. 32, n. 1, p. 1–16, mar. 2006.

PARK, S.; MILLER, K. Random number generators: Good ones are hard to find. Commun. **Communications of the ACM**, [S.1.] v. 31, p. 1192–1201, New York, ago. 1988.

PITON-GONÇALVES, J.; ALUISIO, S. M. A. Teste adaptativo computadorizado multidimensional com propósitos educacionais: princípios e métodos. **Ensaio**: Avaliação e Políticas Públicas em Educação, v. 23, p. 389-414, Rio de Janeiro, jun. 2015.

RABIN, M. O. Probabilistic algorithm for testing primality. **Journal of Number Theory**, v. 12, n. 1, p. 127-138, USA, 1980.

RIVEST, R. L.; SHAMIR, A.; ADLEMAN, L. A method for obtaining digital signatures and public-key cryptosystems. **Communications of the ACM**, v. 21, n. 2, p. 120-126, New York, fev. 1978.

ROHE, M. Randy: a true-random generator based on radioactive decay. In: . [S.l.: s.n.], 2003. Disponível em: < <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.110.9725&rep=rep1&type=pdf>. Acesso em 14 de janeiro de 2020.

SJAFRIE, H. **Introduction to Self-Driving Vehicle Technology**. [S.l.]: Chapman and Hall/CRC, 2019.

SOBOÍ, I. Quasi-monte carlo methods. **Progress in Nuclear Energy**, v. 24, n. 1-3, p. 55-61, Amsterdam, Netherlands, 1990.

SOLOVAY, R. M.; STRASSEN, V. A fast Monte-Carlo test for primality. **SIAM Journal on Computing**, v. 6, n. 1, p. 84–85, Philadelphia, jul. 1977.

SOUZA, G. C. U. I. D. Avaliação de Títulos Conversíveis com Opções de Compra e Venda Implícitas em Contrato. Tese (Doutorado em Engenharia de Produção) - Pontífica Universidade Católica do Rio de Janeiro. Rio de Janeiro, 176, p. 2006.

TAKEFUJI, Y. Reproducibility problems of artificial intelligence inherently come from random numbers. **Science**, v. 369, n. 6388, p. 478, USA, mai. 2018.

WARREN, S. D.; BRANDEIS, L. The Right to Privacy. **Harvard Law Review**, v. 4, n. 5, p. 193-220, Massachusetts Avenue Cambridge, dez. 1890.