Aula 8 - Números Pseudoaleatórios

Prof. Gabriel Rodrigues Caldas de Aquino

Instituto de Computação
Universidade Federal do Rio de Janeiro
gabrielaquino@ic.ufrj.br

Compilado em:
September 19, 2025

Geração de Números Aleatórios em Criptografia

Função importante: geração de números pseudoaleatórios criptograficamente fortes.

- Os **Geradores de Números Pseudoaleatórios (PRNGs)** são essenciais para aplicações de criptografia e segurança.
- Eles produzem sequências de números que parecem aleatórias, mas são geradas a partir de um estado inicial (*seed*).

Importância dos números aleatórios:

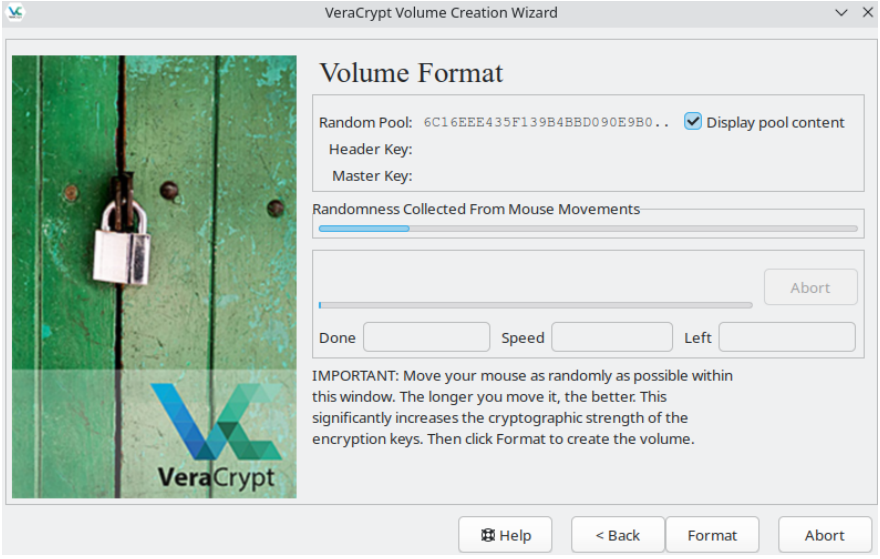
- Garantir **confidencialidade** em criptografia de chave simétrica.
- Produzir **nonces**, IVs e chaves seguras.
- Evitar padrões previsíveis que possam ser explorados por atacantes.
- Fundamentais para diversas aplicações de segurança em redes.

Uso de Números Aleatórios em Segurança de Redes

Função: garantir a segurança em diversos algoritmos criptográficos por meio de números binários aleatórios.

- **Distribuição de chaves e autenticação mútua:**
  - Protocolos cooperativos trocam mensagens para distribuir chaves e autenticar participantes.
  - **Nonces** são usados durante o handshaking para evitar ataques de replay.
  - Números aleatórios nos nonces impedem que atacantes adivinhem ou reutilizem transações antigas.
- **Geração de chave de sessão:**
  - Chaves temporárias para criptografia simétrica, válidas apenas durante uma sessão.
  - Aumenta a segurança limitando a exposição da chave.
- **Geração de chaves para RSA:** Números aleatórios são essenciais para gerar chaves de criptografia assimétrica seguras.
- **Fluxos de bits para cifragem de fluxo:** Algoritmos de cifra de fluxo dependem de números aleatórios para criar sequências criptograficamente seguras.

Exemplo: VeraCrypt tem necessidade de Números Aleatórios



Critérios para Números Aleatórios

Tradicionalmente, a preocupação na geração de uma sequência de números supostamente aleatórios é garantir que a sequência seja estatisticamente aleatória.

Dois critérios principais são usados para validar a aleatoriedade:

- **Distribuição uniforme:** A frequência de ocorrência de uns e zeros deve ser aproximadamente a mesma, garantindo que não haja viés.
- **Independência:** Nenhum valor na sequência pode ser deduzido a partir dos outros; cada elemento é estatisticamente independente dos demais.

Testando Independência de Sequências Aleatórias

Embora existam testes bem definidos para verificar se uma sequência de bits segue uma distribuição específica, como a **uniforme**, não há um teste único para **provar independência**.

Estratégia geral:

- Aplicar diversos testes estatísticos que verificam propriedades relacionadas à independência.
- Se nenhum teste indicar dependência, podemos ter um **alto nível de confiança** de que a sequência é independente.
- A confiança aumenta com o número e a variedade de testes aplicados.

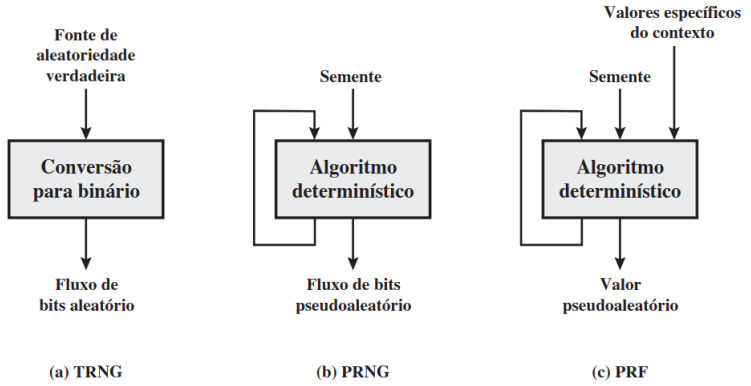
Imprevisibilidade em Números Aleatórios

Em aplicações como autenticação recíproca, geração de chaves de sessão e cifras de fluxo, o requisito principal não é a aleatoriedade estatística, mas a **imprevisibilidade** dos números sucessivos.

- Números verdadeiramente aleatórios: cada elemento é independente e, portanto, imprevisível.
- Limitações dos números verdadeiramente aleatórios: ineficiência e dificuldade de geração.
- Solução comum: algoritmos geradores de números pseudoaleatórios (**PRNGs**) que aparentam aleatoriedade.
- Importante: garantir que um atacante não consiga prever elementos futuros da sequência a partir dos anteriores.

Geradores de números aleatórios e pseudoaleatórios

Figura 7.1 Geradores de números aleatórios e pseudoaleatórios.



TRNG = gerador de número aleatório verdadeiro (true random number generator)
PRNG = gerador de número pseudoaleatório (pseudorandom number generator)
PRF = função pseudoaleatória (pseudorandom function)

TRNGs, PRNGs e PRFs

Geração de números aleatórios em criptografia: normalmente realizada por algoritmos determinísticos.

- **Determinismo:** Algoritmos produzem sequências que não são estatisticamente aleatórias.
- **Números pseudoaleatórios:** Se o algoritmo for bom, a sequência resultante passa em muitos testes de aleatoriedade.
- **Prática aceita:** Apesar de serem determinísticos, PRNGs funcionam "tão bem quanto" números aleatórios em aplicações criptográficas.
- **Analogias estatísticas:** Semelhante a amostras de uma população — resultados aproximam os da população inteira.

TRNG: Geração de Números Aleatórios

Geradores de números aleatórios verdadeiros (TRNG):

- Usam uma fonte efetivamente aleatória, chamada **fonte de entropia**.
- Fontes de entropia podem incluir:
  - Padrões de temporização dos toques de tecla.
  - Movimentos do mouse.
  - Atividade elétrica no disco.
  - Valores instantâneos do clock do sistema.
- A fonte de entropia serve como entrada para um algoritmo que gera saída binária aleatória.
- Pode envolver conversão de sinais analógicos e processamento adicional para eliminar tendências.

TRNG vs PRNG

TRNG – True Random Number Generator

A fonte de entropia é retirada do ambiente físico do computador e pode incluir:

- Padrões de temporização dos toques de tecla
- Atividade elétrica no disco
- Movimentos do mouse
- Valores instantâneos do clock do sistema

A saída é binária e aleatória, podendo envolver conversão de fonte analógica e processamento para reduzir tendências.

TRNG vs PRNG

PRNG – Pseudo-Random Number Generator

Toma como entrada um valor fixo, chamado semente, e produz uma sequência de bits determinística.

- A semente frequentemente é gerada por um TRNG.
- Bits adicionais podem ser produzidos realimentando parte da saída como entrada.
- O fluxo de saída é totalmente determinado pela semente e pelo algoritmo, permitindo reprodução completa se ambos forem conhecidos.

Formas de PRNGs e PRFs

Gerador de Número Pseudoaleatório (PRNG)

Um algoritmo usado para produzir uma sequência de bits tão grande quanto necessário.

- Aplicações típicas: entrada para cifras de fluxo simétricas.
- Pode gerar longas sequências determinísticas a partir de uma semente inicial.
- Consulte a Figura 3.1a para ilustração do fluxo.

Função Pseudoaleatória (PRF)

Produz uma cadeia de bits pseudoaleatória com comprimento fixo.

- Aplicações típicas: geração de chaves de encriptação simétrica e nonces.
- Entrada: semente + valores específicos do contexto (ex.: ID de usuário ou ID de aplicação).

Requisitos de um PRNG/PRF em Criptografia

- O requisito básico: um adversário que não conhece a semente não deve ser capaz de determinar a sequência pseudoaleatória.
- Exemplo em cifras de fluxo: conhecer o fluxo pseudoaleatório permite recuperar o texto claro a partir do texto cifrado.
- Exemplo em PRFs: uma semente de 128 bits e valores de contexto geram uma chave secreta de 128 bits para encriptação simétrica. Saída não suficientemente aleatória facilita ataques por força bruta.
- Implicações: a segurança da saída depende de aleatoriedade, imprevisibilidade e das características da semente.

Aleatoriedade em PRNGs

- O fluxo de bits gerado por um PRNG deve **parecer aleatório**, mesmo sendo determinístico.
- Nenhum teste isolado pode provar aleatoriedade; aplica-se uma **sequência de testes**.
- Segundo o NIST SP 800-22, três características devem ser avaliadas:
  - **Uniformidade**: zeros e uns ocorrem com probabilidade $1/2$; o número esperado de zeros (ou uns) é $n/2$, onde n é o tamanho da sequência.
  - **Escalabilidade**: subsequências extraídas aleatoriamente também devem passar nos testes de aleatoriedade.
  - **Consistência**: o comportamento do gerador deve ser coerente entre diferentes sementes.
- Veja os testes do NIST
- Software de testes do NIST
- Não é adequado testar um PRNG usando apenas uma única semente ou um TRNG usando apenas uma saída física.

Testes de Aleatoriedade do NIST SP 800-22

- O SP 800-22 lista 15 testes separados para avaliar a aleatoriedade de um PRNG.
- Aqui estão três exemplos e seus propósitos:
  - **Teste de frequência**: verifica se o número de uns e zeros na sequência é aproximadamente o esperado para uma sequência verdadeiramente aleatória.
  - **Teste de rodadas**: avalia o número total de rodadas (sequências consecutivas de bits idênticos limitadas por bits opostos) para ver se correspondem ao esperado.
  - **Teste da estatística universal de Maurer**: mede a distância entre padrões correspondentes; identifica se a sequência é significativamente comprimível e, portanto, não aleatória.
- Esses testes fornecem uma visão geral do comportamento estatístico das sequências pseudoaleatórias.

Imprevisibilidade de Números Pseudoaleatórios

Um fluxo de números pseudoaleatórios deve exibir duas formas de imprevisibilidade:

- **Imprevisibilidade direta**: se a semente é desconhecida, o próximo bit da sequência deve ser imprevisível, mesmo conhecendo todos os bits anteriores.
- **Imprevisibilidade inversa**: não é viável determinar a semente a partir de quaisquer valores gerados. Nenhuma correlação entre a semente e os bits gerados deve ser aparente.

- O mesmo conjunto de testes de aleatoriedade também verifica a imprevisibilidade.
- Se a sequência parece aleatória, não é possível prever bits futuros nem deduzir a semente a partir da sequência conhecida.

Requisitos da Semente para PRNGs Criptográficos

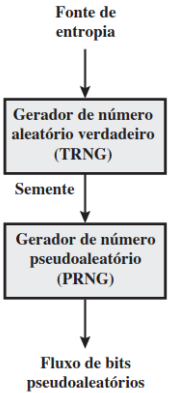
Para aplicações criptográficas, a semente de um PRNG deve ser segura:

- **Imprevisível**: se o adversário puder deduzir a semente, poderá reproduzir toda a saída do PRNG.
- Normalmente gerada por um **TRNG**, garantindo que a semente seja aleatória ou pseudoaleatória.

- Em cifras de fluxo, o TRNG não é prático para gerar o fluxo de chave completo; o PRNG permite transmitir apenas a chave curta com segurança.
- Para funções pseudoaleatórias (PRFs), o TRNG fornece a semente, e a PRF gera os bits de saída para eliminar possíveis vieses do TRNG.
- O TRNG pode ter limitações de velocidade; o PRNG permite gerar números suficientes para a aplicação.

PRNG e a seed

Figura 7.2 Geração de entrada de semente ao PRNG.



Projeto de Algoritmos de PRNG Criptográficos

PRNGs criptográficos podem ser classificados em duas categorias principais:

- **Algoritmos de propósito especial**: projetados especificamente para gerar fluxos de bits pseudoaleatórios. Alguns são usados para várias aplicações de PRNG; outros são dedicados a cifras de fluxo, como o **RC4**.
- **Algoritmos baseados em algoritmos criptográficos existentes**: utilizam propriedades de algoritmos criptográficos para produzir aleatoriedade.

Três técnicas gerais para gerar PRNGs criptograficamente fortes:

- **Cifras de bloco simétricas**
- **Cifras assimétricas**
- **Funções de hash e códigos de autenticação de mensagem**

Essas técnicas podem ser combinadas e reutilizadas de forma prática, aproveitando algoritmos já existentes em sistemas de encriptação e autenticação.

Fontes de Entropia em TRNGs

Geradores de números aleatórios verdadeiros (TRNGs) utilizam fontes não determinísticas:

- Medem processos naturais imprevisíveis, como:
  - Detectores de pulso de radiação ionizante.
  - Tubos de descarga de gás.
  - Capacitores com escape.
- Exemplos comerciais e de pesquisa:
  - Chip da Intel: captura ruído térmico, amplificando voltagem de resistores não condutíveis.
  - **LavaRnd**: projeto de código aberto que gera números verdadeiramente aleatórios usando câmeras baratas e CCD saturado como fonte caótica. Link
- O software processa as amostras e produz números aleatórios em vários formatos.

Fontes de Aleatoriedade em Computadores (RFC 4086)

Possíveis fontes de aleatoriedade para gerar sequências verdadeiramente aleatórias:

- **Entrada de som/vídeo:**
  - Microfone sem entrada ou câmera tampada produz ruído térmico.
  - Digitalização fornece bits aleatórios de qualidade razoável.
- **Unidades de disco:**
  - Pequenas flutuações aleatórias na rotação devido à turbulência do ar.
  - Medições de tempo de busca geram dados aleatórios, embora correlacionados.
  - Processamento adequado pode extrair excelentes bits aleatórios (ex.: 100 bits/min em discos antigos).
- Serviço on-line random.org que pode oferecer sequências aleatórias pela Internet.
- Exemplo de TCC no tema de geração de números aleatórios: Link

Geradores de números aleatórios e pseudoaleatórios

Tabela 7.5 Comparação entre PRNGs e TRNGs.

	Geradores de números pseudoaleatórios (PRNGs)	Geradores de números aleatórios verdadeiros (TRNGs)
Eficiência	Muito eficientes	Geralmente ineficientes
Determinismo	Determinísticos	Não determinísticos
Periodicidade	Periódicos	Aperiódicos

PRNGs: Números Pseudoaleatórios Determinísticos

Características principais dos PRNGs:

- **Eficiência:** podem gerar muitos números rapidamente, adequado para aplicações que exigem grandes quantidades de números pseudoaleatórios.
- **Determinismo:** a mesma sequência pode ser reproduzida se a condição inicial (semente) for conhecida.
- **Periodicidade:** as sequências eventualmente se repetem, mas PRNGs modernos possuem períodos tão longos que são irrelevantes para a maioria das aplicações.

TRNGs: Números Aleatórios Verdadeiros

Características principais dos TRNGs:

- **Eficiência:** geralmente mais lentos que PRNGs, podendo ser um gargalo em aplicações que exigem muitos números aleatórios por segundo.
- **Não determinísticos:** a mesma sequência não pode ser reproduzida de forma garantida.
- **Sem periodicidade:** não possuem ciclo repetitivo, diferentemente dos PRNGs.
- **Aplicações críticas:** úteis em contextos onde imprevisibilidade máxima é necessária, como criptografia bancária ou nacional.

Propensão em TRNGs e Algoritmos Antipropensão

Propensão: um TRNG pode gerar uma saída tendenciosa, com mais uns ou zeros.
Soluções:

- Aplicação de **algoritmos antipropensão** para reduzir ou eliminar a tendência.
- Uso de **funções de hash** (MD5, SHA-1) para misturar blocos de bits:
  - Blocos de $m \geq n$ bits de entrada processados para gerar n bits de saída.
  - Mistura de entradas de diferentes fontes de hardware.
- Sistemas operacionais oferecem mecanismos embutidos:
  - Linux: combina atividade do mouse/teclado, E/S de disco e interrupções. A saída aleatória passa por SHA-1 antes de ser fornecida.
  - Leia

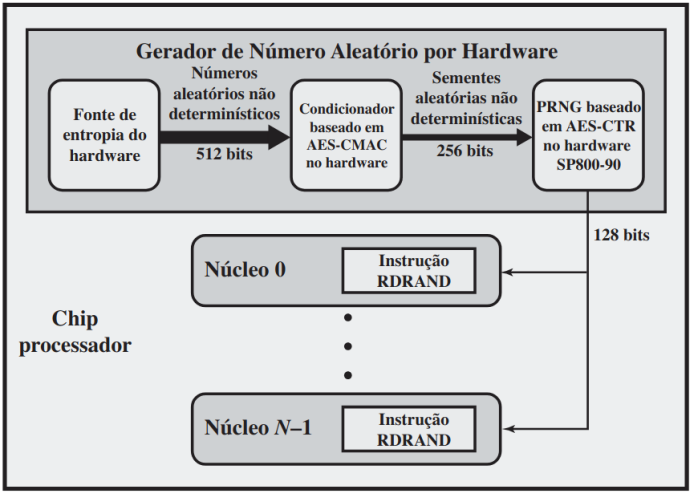
Gerador de Números Aleatórios Digitais da Intel (DRNG)

Contexto: TRNGs tradicionais eram usados apenas para gerar pequenas quantidades de bits aleatórios devido à baixa taxa de produção.
Intel DRNG: primeiro TRNG comercial com taxa de produção comparável a PRNGs, disponível em chips multicore desde 2012.
Destaques:

- **Implementação totalmente em hardware:** maior segurança e velocidade comparada a soluções baseadas em software.
- **Integração no chip multicore:** elimina atrasos de E/S encontrados em outros TRNGs.

Intel DRNG

Figura 7.9 Chip processador da Intel com gerador de números aleatórios.



Visão Geral do Intel DRNG

O DRNG é uma arquitetura de **três estágios**:

1. **Fonte de Entropia**: Gera bits aleatórios a partir do ruído térmico.
2. **Condicionador**: Remove qualquer viés ou correlação residual.
3. **PRNG**: Expande a aleatoriedade para maior throughput.

Estágio 1: Fonte de Entropia (O Núcleo Aleatório)

- **Núcleo**: Dois inversores (portas NOT).
- Possui dois estados estáveis lógicos.
- Pulsos de *clock* forçam ambos para um **estado metaestável** indeterminado.
- O **ruído térmico** aleatório dentro dos transistores decide para qual estado estável o circuito irá decair.
- Este decaimento é fundamentalmente **imprevisível**.

Desempenho

Gera bits a uma taxa de **4 Gbps** (Gigabits por segundo). A saída é colhida em blocos de **512 bits**.

Estágio 2: Condicionamento com CMAC

- **Problema**: A saída do estágio 1 pode ter **viés** ou ter **correlações** sutis.
- **Solução**: Um condicionador criptográfico.
- **Função**: CBC-MAC (CMAC), conforme padrão NIST SP 800-38B.

Como funciona?

- Os 512 bits do Estágio 1 são criptografados em modo **CBC** usando o algoritmo AES.
- Apenas o **último bloco cifrado** (o MAC) é tomado como saída.
- Esta etapa distilaä entropia, produzindo **256 bits** verdadeiramente aleatórios e não enviesados por bloco.

Estágio 3: Geração em Alta Velocidade (PRNG)

- **Problema**: Mesmo a 4 Gbps, a entropia pura não é suficientemente rápida para todas as aplicações modernas.
- **Solução**: Usar os bits aleatórios do Estágio 2 como **semente** para um PRNG.

Algoritmo: CTR_DRBG

Counter Mode Deterministic Random Bit Generator

- **Semente**: Os 256 bits do Estágio 2.
- **Funcionamento**: Criptografa um contador incremental usando AES.
- **Saída**: Gera números pseudoaleatórios de **128 bits**.
- **Segurança**: Sem a semente, é computacionalmente inviável prever a saída.

Vantagem

A partir de uma única semente de 256 bits, o PRNG pode gerar **muitos** blocos de 128 bits, ultrapassando a taxa da fonte de entropia ($>$**3 Gbps**). Um limite de **511 amostras** é definido por semente para garantir a segurança antes de uma nova ressementeação.

Interface com o Software: A Instrução RDRAND

- A saída final do DRNG (seja do Estágio 2 ou 3) é disponibilizada para todos os núcleos do processador.
- Acesso via uma instrução de assembly dedicada: **RDRAND**.

Instrução RDRAND

`RDRAND reg`

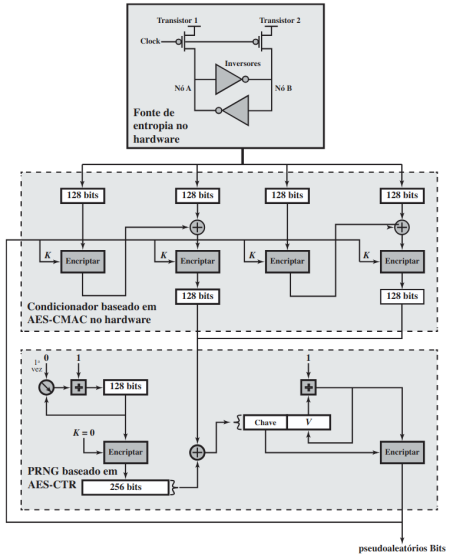
onde `reg` pode ser um registrador de 16, 32 ou 64 bits (`AX`, `EAX`, `RAX`).

- A instrução busca um valor aleatório do tamanho solicitado.
- Define a flag de *carry* (CF=1) em caso de sucesso.
- Software deve verificar o *carry* para garantir que obteve um valor aleatório válido.

Leitura interessante - Link da Intel

Intel DRNG: Estrutura lógica

Figura 7.10 Estrutura lógica do DRNG da Intel.



Código executando o rdrand

```
section .text
    global _start

_start:
    rdrand rcx          ; Bota valor aleatorio no registrador RCX - 64 bits
    jnc .exit           ; Vai pular se NÃO tiver o carry flag (CF = 0)

    ; Escreve os 64 bits diretamente na saída padrao (terminaL)
    push rcx            ; coloca os 64 bits na stack
    mov rax, 1          ; Fala qual e a operacao - sys_write
    mov rdi, 1          ; Fala aonde que vai escrever - stdout
    mov rsi, rsp        ; Fala onde que é pra ler - aponta leitura para os dados na stack (rsp é o stack pointer)
    mov rdx, 8          ; Fala a quantidade pra ler - 8 bytes (64 bits)
    syscall

    add rsp, 8          ; Joga o stack pointer 8 bytes para cima - ou seja, limpa a stack

.exit:
    mov rax, 60         ; Fala qual que e a operacao - no caso sys_exit
    xor rdi, rdi        ; Bota zero no rdi - exit(0)
    syscall
```

Compilando o código com rdrand

```
$nasm -f elf64 rdrand_bin.asm -o rdrand_bin.o
$ld rdrand_bin.o -o rdrand_bin
$ ./rdrand_bin | hexdump -C
```