# GBC083 - Segurança da Informação
# Aula 3 - Pseudo-aleatoriedade

# Geração de chaves aleatórias

- Ao descrever o algoritmo OTP, assumimos que temos acesso a bits uniformemente aleatórios
- Onde encontrar esses números?
- Geração de número aleatórios e pseudo-aleatórios

# O que significa algo aleatório?

- O que significa uniforme?
- Qual string é aleatória?
  - 01010101010101010101
  - 01011111010011100110
  - 00000000000000000001
- Se geramos string uniforme, cada uma dessas tem probabilidade igual de aparecer

# O que significa algo aleatório?

- Aleatoriedade não é propriedade de uma string mas de uma distribuição
- Uma distribuição em string de $n$ bits é uma função $D : \{0,1\}^n \to [0,1]$ tal que $\sum_x D(x) = 1$
  - A distribuição uniforme $U_n$ é tal que $P(U_n = x) = 2^{-n}$ para todo $x \in \{0,1\}^n$

# O que significa algo pseudo-aleatório?

- Informal: algo que não pode ser diferenciado de algo uniforme
- Pseudo-aleatoriedade é uma propriedade de uma distribuição

# Pseudo-aleatoriedade

- Escolher uma distribuição $D$ de string de $n$ bits
  - $x \leftarrow D$ significa "amostrar $x$ de acordo com $D$"
- Historicamente, $D$ era considerado pseudo-aleatório se passava por um conjunto extensivo de testes estatísticos
  - $P_{x \leftarrow D}(\text{1o. bit de x é 1}) \approx 0.5$
  - $P_{x \leftarrow D}(\text{paridade de x é 1}) \approx 0.5$
  - $P_{x \leftarrow D}(A_i(x) = 1) \approx P_{x \leftarrow U_n}(A_i(x) = 1)$ para $i \in 1 \ldots 20$

# Pseudo-aleatoriedade

- Mas testes não representa todo cenário de atacantes
  - Testes de atacantes são imprevisíveis
- Definição criptográfica de pseudo-aleatoridade:
  - $D$ é pseudo-aleatório se passa por todos testes estatísticos eficientes

# Pseudo-aleatoriedade concreta

- Seja $D$ uma distribuição de string de $n$ bits
- $D$ é $(t, \epsilon)$-pseudo aleatória se para todo $A$ rodando em tempo $\leq t$,

$$|P_{x \leftarrow D}(A(x) = 1) - P_{x \leftarrow U_n}(A(x) = 1)| \leq \epsilon$$

# Pseudo-aleatoriedade assintótica

- Parâmetro de segurança $n$ e polinomial $p$
- Seja $D_n$ uma distribuição sobre strings de $p(n)$ bits
- Pseudo-aleatoriedade é uma propriedade de uma sequência de distribuições $\{D_n\} = \{D_1, D_2, \ldots\}$

# Pseudo-aleatoriedade assintótica

- $\{D_n\}$, onde $D_n$ é uma distribuição sobre strings de $p(n)$ bits, é pseudo-aleatória se para todos atacantes TPP $A$ existe uma função negligível $\epsilon$ tal que

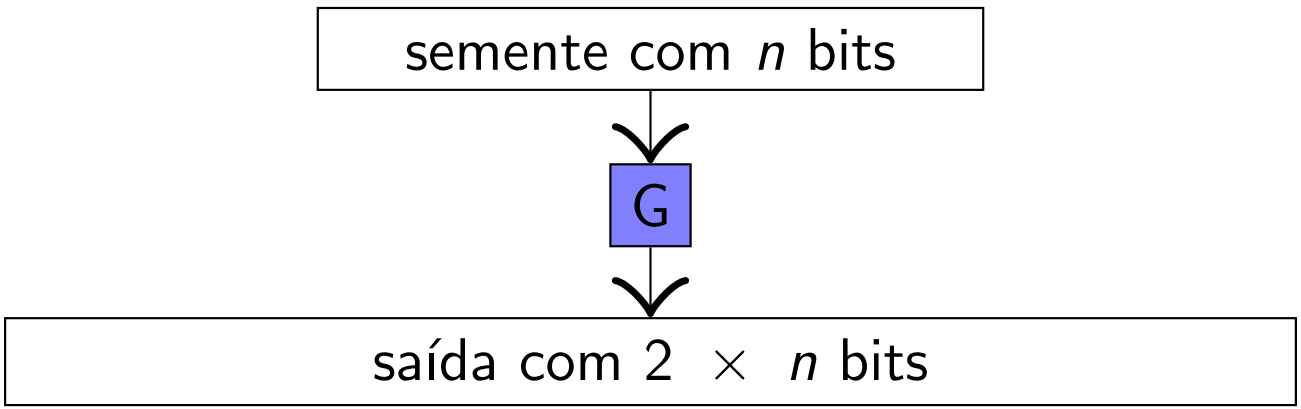
$$|P_{x \leftarrow D_n}(A(x) = 1) - P_{x \leftarrow U_{p(n)}}(A(x) = 1)| \leq \epsilon(n)$$

# Geradores Pseudo-Aleatórios (GPA)

- Um GPA é um algoritmo eficiente e determinístico que transforma uma "semente" pequena e uniforme em uma sequência mais longa e pseudo-aleatória
  - Útil quando temos uma quantidade pequena de bits aleatórios e queremos muitos bits pseudo-aleatórios
- Em inglês: Pseudo-Random Number Generator (PNRG)

# Geradores Pseudo-Aleatórios (GPA)

- Seja $G$ um algoritmo polinomial determinístico
- $G$ é expansível se $|G(x)| = p(|x|) > |x|$



# Geradores Pseudo-Aleatórios (GPA)

- Uma função expansível $G$ define um conjunto de distribuições $D = \{D_1, D_2, D_{...}, D_{p(n)}, \ldots\}$
  - onde $D_{p(n)}$ é distribuição em strings de $p(n) > |x| = n$ bits definida ao escolher $x \leftarrow U_n$ (semente) e usar o GPA $G(x)$
  - $D_{p(n)}$ é definida formalmente por

$$
\begin{aligned}
P_{D_{p(n)}}(y) &= P_{U_n}(\{x : G(x) = y\}) = \sum_{x:G(x)=y} P_{U_n}(x) \\
&= \sum_{x:G(x)=y} 2^{-n} \\
&= |\{x : G(x) = y\}|/2^n
\end{aligned}
$$

# Geradores Pseudo-Aleatórios (GPA)

- Seja $G$ um algoritmo polinomial determinístico
- $G$ é expansível: $|G(x)| = p(|x|) > |x|$
- $G$ é um gerador pseudo-aleatório (GPA) se $\{D_n\}$ for pseudo-aleatório

# Geradores Pseudo-Aleatórios (GPA)

- $G$ é um gerador pseudo-aleatório se $\{D_n\}$ for pseudo-aleatório
  - Ou seja, para qualquer atacante polinomial $A$ existe função negligível $\epsilon$ tal que

$$|P_{x \leftarrow D_n}(A(G(x)) = 1) - P_{y \leftarrow U_{p(n)}}(A(y) = 1)| \leq \epsilon(n)$$

  - Nenhum atacante eficiente $A$ consegue discernir se é uma dada $G(x)$ ou se é de fato uma string uniforme $y$

# Existem Geradores Pseudo-Aleatórios (GPA)?

- Não se sabe ainda... :-(
- Na prática, assume-se que algumas funções $G$ são GPA
- Existem GPA se usarmos outras definições mais fracas

# Diferença entre um Gerador Aleatório e um Pseudo-Aleatório

- Supor que o gerador pseudo aleatório (GPA) dobra o tamanho da semente: $l(n) = 2n$
- Tamanho de espaço de strings binárias aleatórias com tamanho $2n$: $2^{2n}$
- Tamanho do maior espaço que o GPA pode alcançar: $2^n$
- O GPA pode gerar apenas uma fração de $\frac{1}{2^n}$ do espaço verdadeiramente aleatório.

# Ataque de força-bruta a um GPA (parte 1)

- Um atacante $D$ quer usar força-bruta para verificar se $w \in \{0,1\}^n$ é aleatória ou pseudo-aleatória de um GPA $G(\cdot)$ com $l(n) = 2n$
- Atacante gera cada $y = G(s \in \{0,1\}^n)$
- Atacante $D$, quando aplicado a $G$, retorna 1 se bem sucedido: $D(G(s)) = 1$

# Ataque de força-bruta a um GPA (parte 2)

- Se $w$ foi gerado por $G(\cdot)$ então atacante encontra $w = y$ com probabilidade 1, ou seja,

$$P(D(G(s)) = 1) = 1$$

- Se $w = r$ foi gerado tal que $r \in U^{2n}(\cdot)$, então a chance de $r \in G(\cdot)$ é:

$$P(D(r) = 1) = \frac{1}{2^n}$$

- Então a margem de sucesso é

$$|P(D(r) = 1) - P(D(G(s)) = 1)| = 1 - 2^{-n} > \epsilon(n)$$

# Geração de números aleatórios na prática

- Dois passos
  - Continuamente manter uma reserva (um "pool") de alta-entropia, ou seja, dados imprevisíveis
  - Quando bits aleatórios são requisitados, processar esses dados para gerar uma sequência de bits uniforme e independente
- Detalhes dependem do sistema
  - Linux: SHA1
  - iOS: algoritmo de Yarrow (milefólio)
  - FreeBSD: Fortuna
- Bibliotecas de criptografia

# Passo 1 – eventos coletados (Linux)

- Coletar uma reserva (pool) de dados de alta entropia de até 4096 bits
  - `/proc/sys/kernel/random/entropy_avail` – índice de aleatoriedade no pool de entrada
- Entradas externas:
  - Atrasos entre eventos de redes de computadores
  - Tempos de acesso a disco rígido
  - Medições de tempo de interrupções
  - Movimentos de teclado e mouse
  - Hardware de geração de números aleatórios

# Passo 1 – processando eventos (Linux)

- Para cada evento de entrada, processar 3 números de 32 bits:
  - `num`: número identificador do evento (se teclado, `keycode`)
    - Teclado: entropia máxima é 8 bits
    - Mouse: 12 bits
    - Disco rígido: 3 bits
    - Interrupção: 4 bits
  - `cycle`: conteúdo do registrador de contagem de ciclos da CPU
    - $\approx$ 14.8 bits de entropia
  - `jiffies`: contador do kernel para o número de interrupções de relógio desde último boot
    - $\approx$ 3.4 bits de entropia

    Dados de "The Linux Pseudorandom Number Generator Revisited" `http://eprint.iacr.org/2012/251.pdf`
- Produz em geral menos que 30 bits de entropia (aleatoriedade)

# Passo 2 (Linux)

- Passo 2: processar pool de dados de alta-entropia.
- Mantém 2 pools de até 1024 bits aleatórios de saída:
  - `/dev/random` – bloqueia na falta de aleatoriedade
  - `/dev/urandom` ou `get_random_bytes()` (`linux/random.h`) – continua produção de números pseudo-aleatórios
    - `http://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/drivers/char/random.c`
- Misturar dados recebidos com pool atual usando a função CRC:
  - Rápido e seguro se: 1) aleatoriedade não vem de um atacante; 2) difícil de atacante externo medir.
- Obter número aleatório com uma função hash SHA1 do pool de bits aleatórios
  - Uniformemente aleatório se a quantidade de bits gerada for menor que a entropia do pool
  - Reduz número de bits de entropia disponíveis ao fornecer bits aleatórios

# Histórias reais sobre aleatoriedade

- Padrão NIST SP800-90A – Dual_EC_DRBG (baixar)
- Histórias de horror:
  - `https://nakedsecurity.sophos.com/2011/11/07/randomness-in-cryptography-the-devils-in-the-details/`
  - NSA backdoor in the Dual_EC_DRBG PRNG
  - The many flaws in DUAL_EC_DRBG PRNG

# Próximos assuntos

- Pseudo One-Time Pad
- Prova de segurança computacional