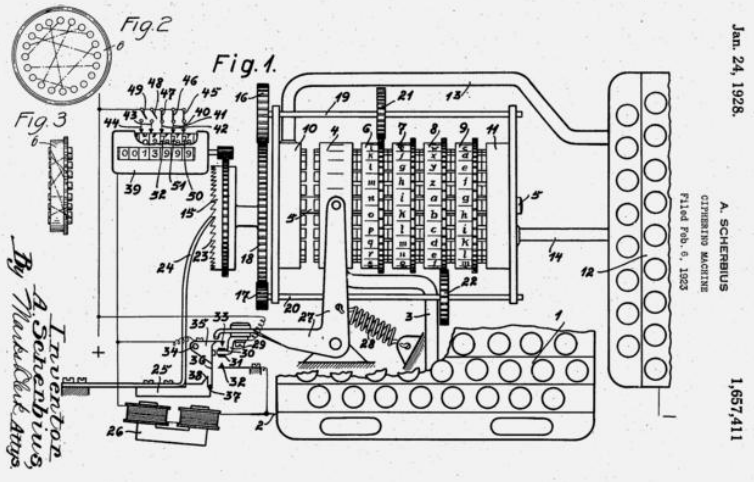
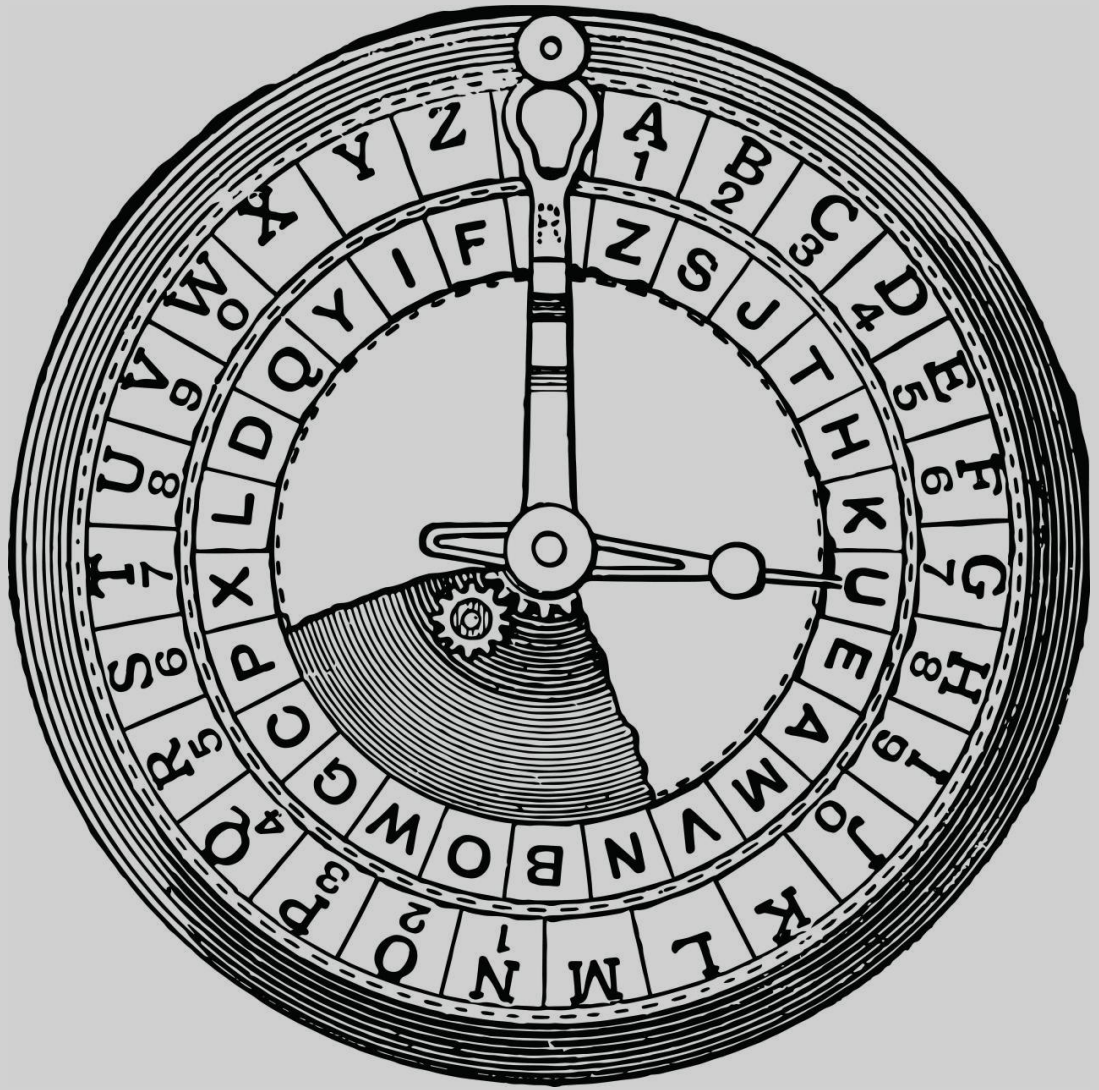
Primitivas Criptográficas

Fundamentos da Criptografia

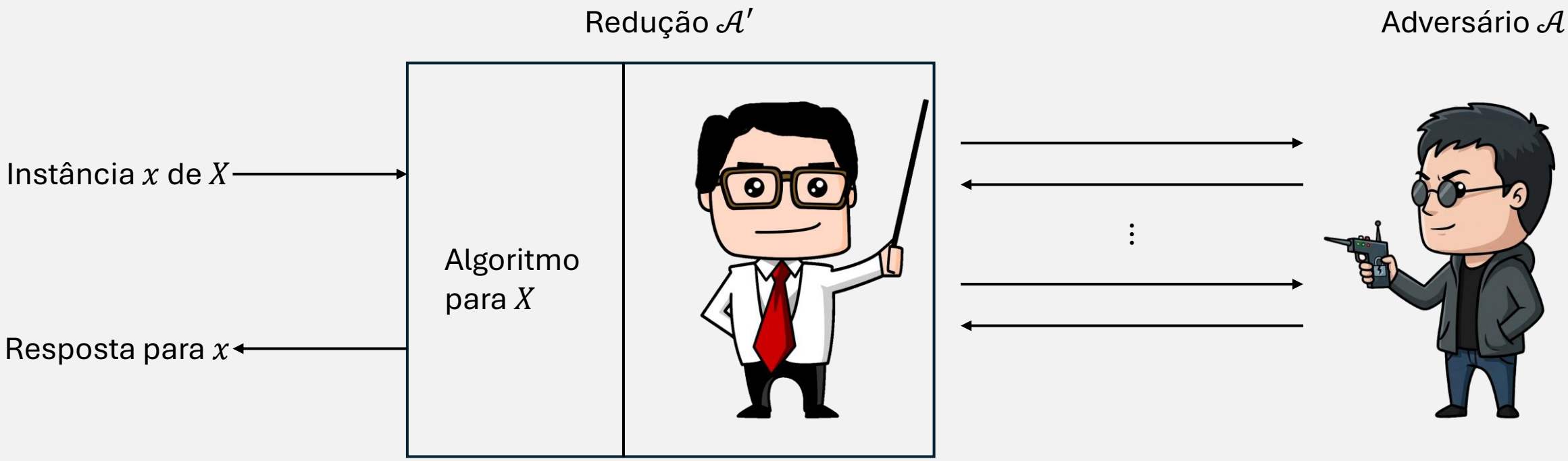
Nicollas Mocelin Sdroievski



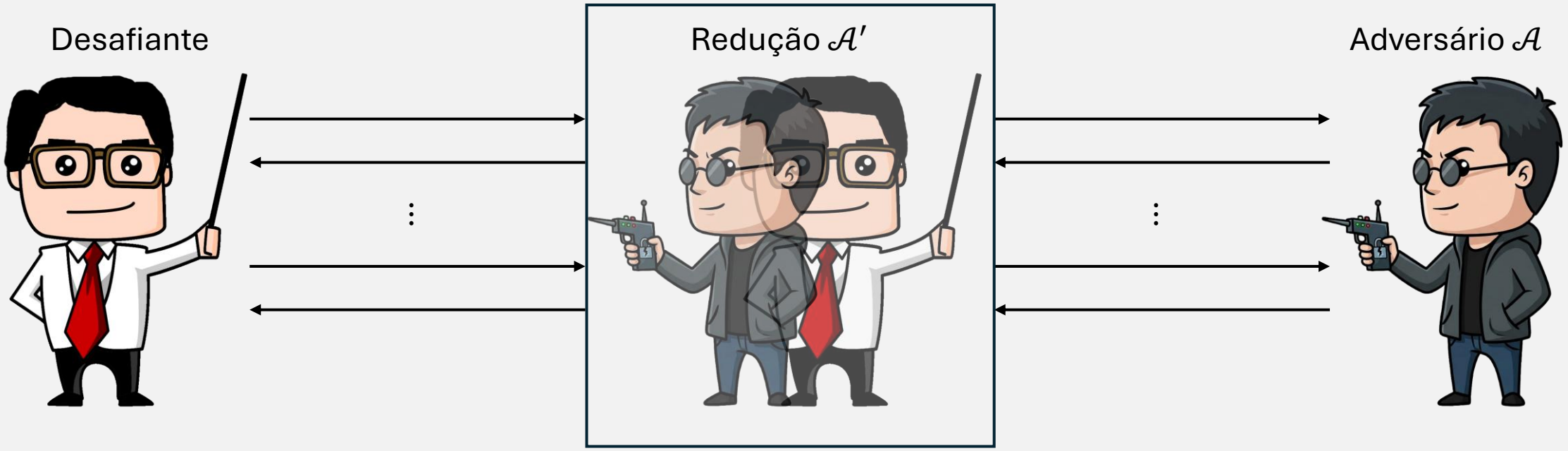
Na última aula



Reduções


Redução $\mathcal{A}'$
Adversário $\mathcal{A}$
Instância x de X
Resposta para x
Algoritmo para X

Reduções entre protocolos



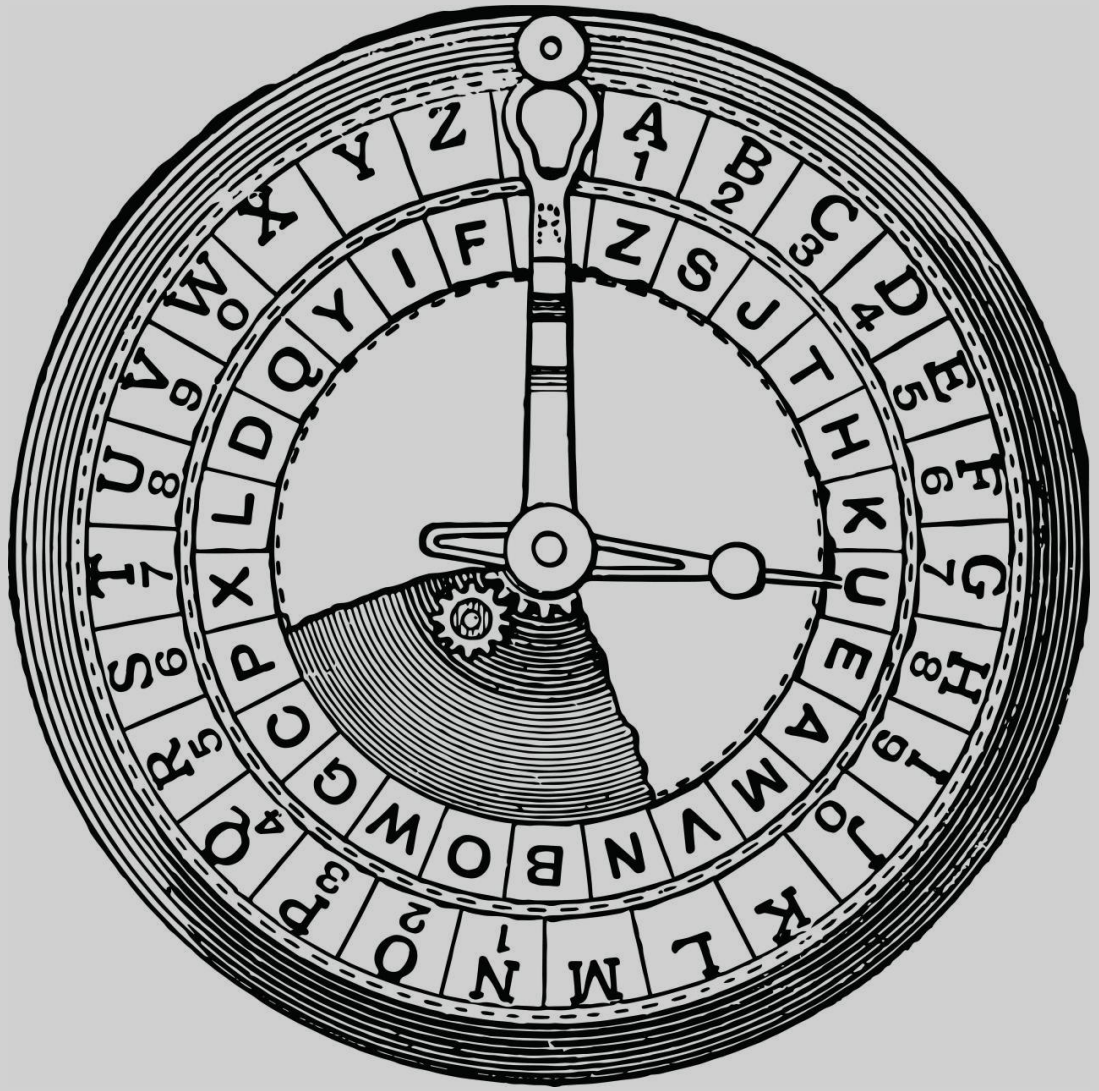
Onde terminam as reduções?

- Quero provar que Π é seguro
- Assumindo que Γ_1 é seguro, provo que Π é seguro
- Quero provar que Γ_1 é seguro
- Assumindo que Γ_2 é seguro, provo que Γ_1 é seguro
- $\vdots$
- Assumindo uma **primitiva criptográfica**, provo que Γ_i é seguro

Primitivas Criptográficas

- Suposições básicas (quase) universalmente aceitas como verdadeiras
  - Simples
  - Extensivamente testadas
  - Em muitos casos, mínimas
- Exemplos
  - Funções unidirecionais ou funções de mão-única
  - Geradores pseudoaleatórios
  - Funções pseudoaleatórias (mais pra frente)

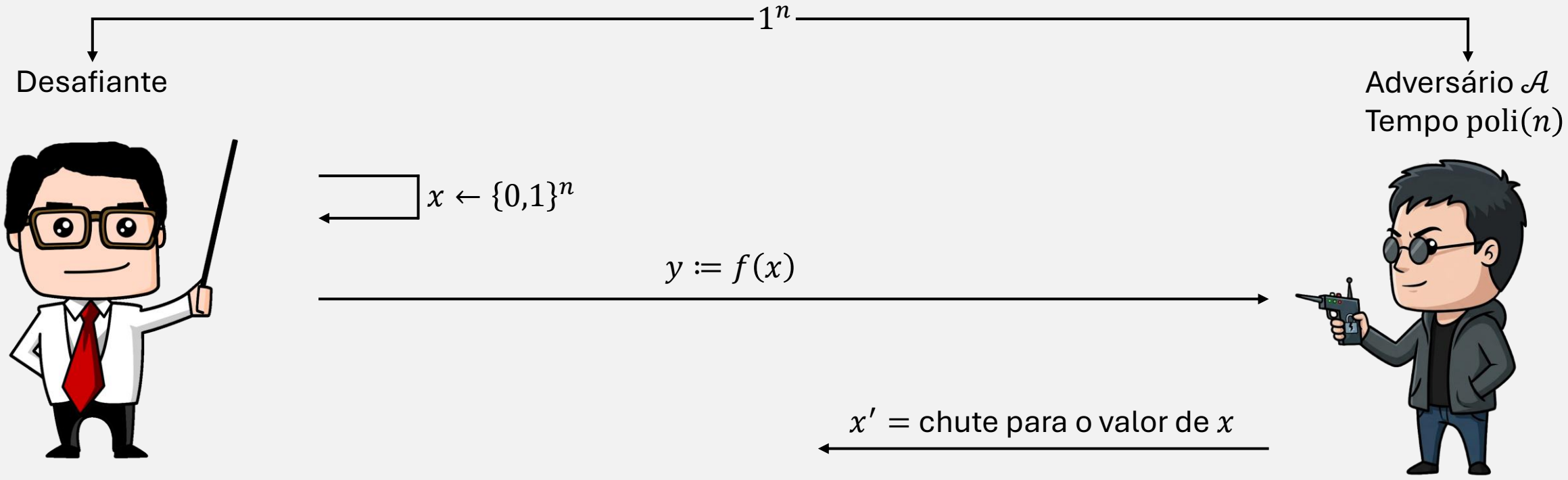
Funções unidirecionais



Ideia geral

- Uma das primitivas mais simples
- E a mais extensivamente testada
- Uma função unidirecional (OWF) é uma função f tal que
  - Computar $f(x)$ deve ser fácil
  - Dado y, encontrar x' tal que $f(x') = y$ (ou seja, inverter f) é difícil
- Algumas perguntas
  - Que tipo de dificuldade queremos ao tentar inverter f?
    - Dificuldade de caso médio
  - Como definir o conceito de função unidirecional?
    - Com um experimento!

Experimento $\text{Invert}_{\mathcal{A},f}(n)$



- A saída do experimento $\text{Invert}_{\mathcal{A},f}(n)$ é igual a 1 se $f(x') = y$, e 0 caso contrário

Definição

- **Definição 7.1.** Uma função $f : \{0,1\}^* \to \{0,1\}^*$ é **unidirecional** se as duas condições a seguir são respeitadas:

1. **(Fácil de computar.)** Existe um algoritmo polinomial M_f que computa f; isto é, $M_f(x) = f(x)$ para todo x.
2. **(Difícil de inverter.)** Para todo algoritmo probabilístico polinomial $\mathcal{A}$, existe uma função negligenciável $\mathrm{negl}(n)$ tal que

$$\Pr\left[\mathrm{Invert}_{\mathcal{A},f}(n) = 1\right] \leq \mathrm{negl}(n)$$

Observações

- A probabilidade $\Pr\left[\text{Invert}_{\mathcal{A},f}(n) = 1\right]$ é sobre x e a aleatoriedade de $\mathcal{A}$
- Podem existir valores de x tal que $f(x)$ é "fácil" de inverter
  - Porém a fração desses x deve ser negligenciável
- Imagine que para $X \in \{0,1\}^n$ com $|X| \geq 2^n/n$, $\mathcal{A}$ consegue inverter $x \in X$ com probabilidade pelo menos $1/n$. Então:

$$\Pr\left[\text{Invert}_{\mathcal{A},f}(n) = 1\right] =$$

$$= \Pr\left[\text{Invert}_{\mathcal{A},f}(n) = 1 \mid x \in X\right] \cdot \Pr[x \in X] + \Pr\left[\text{Invert}_{\mathcal{A},f}(n) = 1 \mid x \notin X\right] \cdot \Pr[x \notin X]$$

$\geq \frac{1}{n} \cdot \frac{1}{n} + 0 = \frac{1}{n^2}$, que não é negligenciável

Exemplos

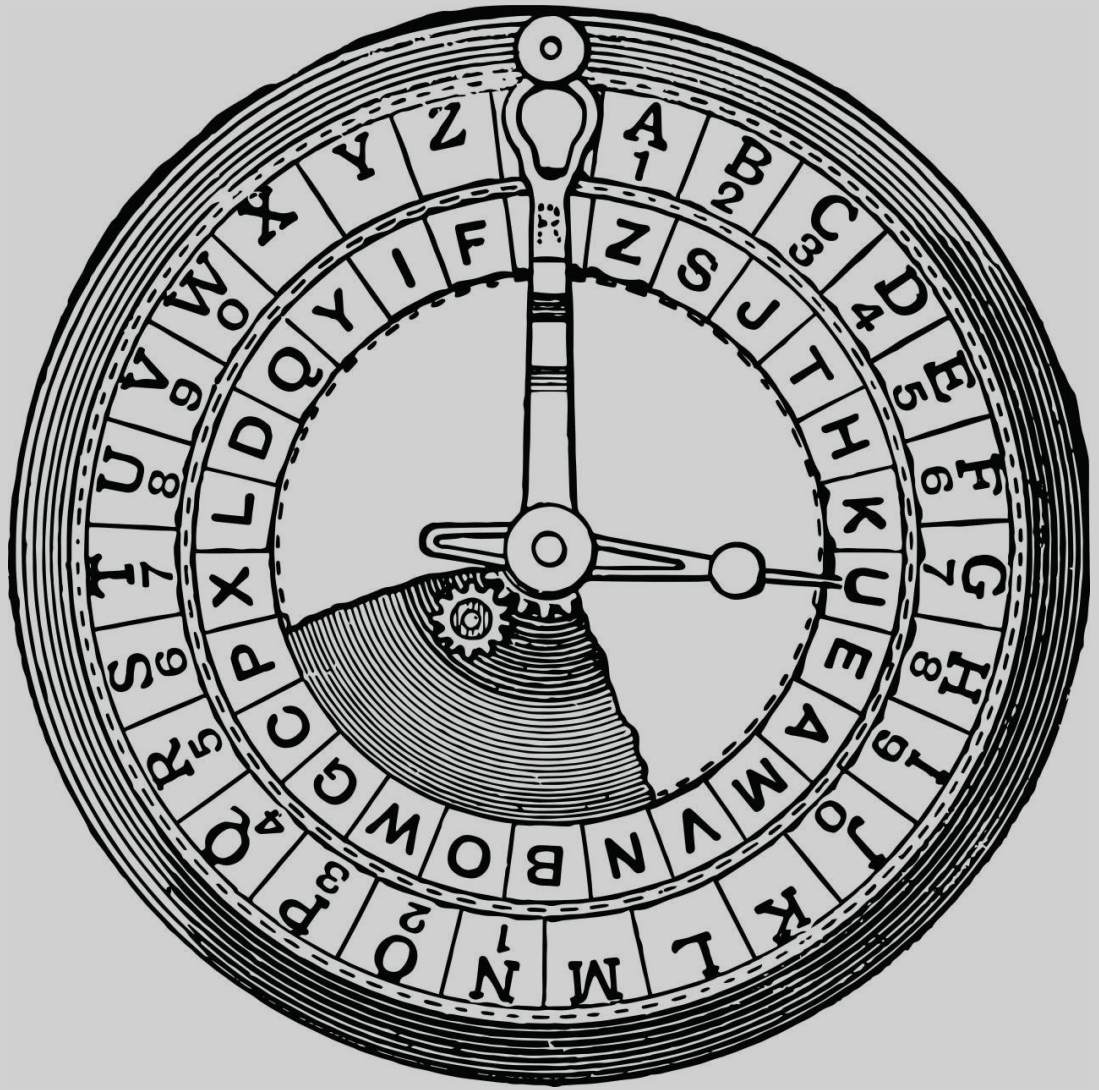
- Multiplicação / Fatoração
  - $f(p, q) = p \cdot q$
  - Para inverter f, temos que fatorar o número dado como entrada
  - p e q tem que ser primos com cerca de $n/2$ bits cada (por que?)

- Exponenciação Modular / Logaritmo Discreto
  - $f_{p,g}(x) = g^x \pmod{p}$
  - Para inverter f, temos que computar o logaritmo discreto (base g e módulo p) do número dado como entrada
  - p deve ser primo e g deve ser um gerador de $\mathbb{Z}_p^*$

Um exemplo um pouco diferente

- Produto Interno / Soma de Subconjuntos
  - $f(a_1, \dots, a_n, x_1, \dots, x_n) = \left(a_1, \dots, a_n, \sum_{i=1}^{n} x_i \cdot a_i \pmod{2^n}\right)$
  - Para inverter, é necessário resolver o problema de soma de subconjuntos com $(a_1, \dots, a_n)$ e algum número t
- Observação
  - Soma de Subconjuntos é NP-completo, mas isso não é suficiente
  - Queremos problemas que são difíceis no caso médio, não apenas no pior caso

Geradores pseudoaleatórios



Ideia geral

- Geradores pseudoaleatórios (PRGs de *pseudorandom generators*):
  - Algoritmos que “esticam” uma string aleatória s (a semente), produzindo uma string maior
- A string produzida apenas “parece” aleatória
- Ideia antiga: desenvolver vários testes e definir como pseudoaleatórios os geradores que passam nestes testes
  - O primeiro bit deve ser 1 com probabilidade aproximadamente $1/2$
  - A paridade dos bits deve ser 1 com probabilidade aproximadamente $1/2$
  - ⋮
- Hoje: o gerador deve passar em **todos** os testes eficientes
  - Testes implementados por algoritmos probabilístico polinomiais

Definição formal

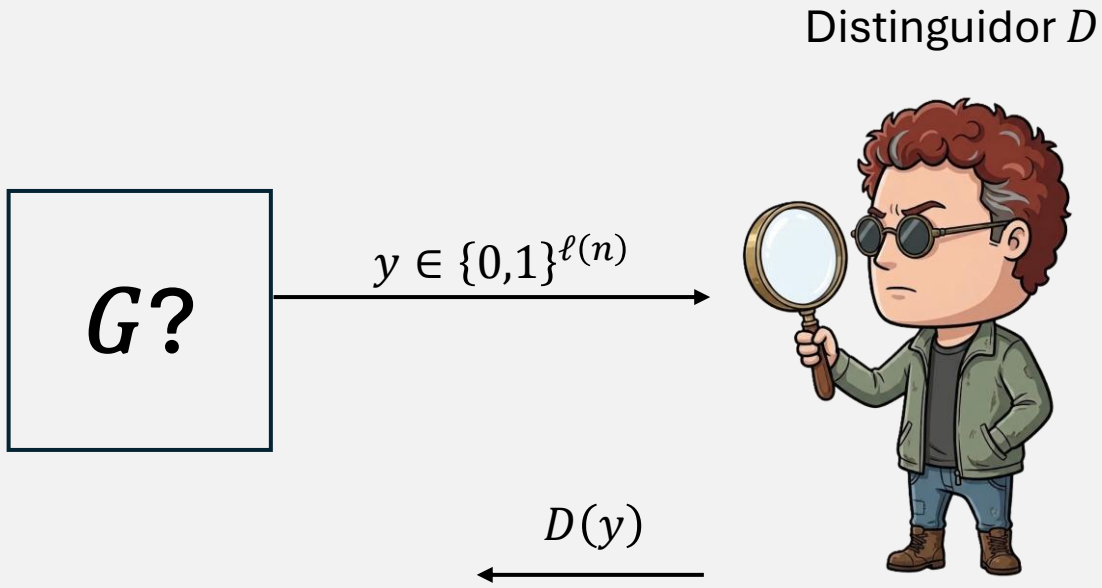
- **Definição 3.14.** Seja ℓ um polinômio e G um algoritmo determinístico polinomial tal que para todo n e $s \in \{0,1\}^n$, $G(s)$ é uma string de tamanho $\ell(n)$. Dizemos que G é um gerador pseudoaleatório se o seguinte é verdade:

1. **(Expansão.)** Para todo n, vale que $\ell(n) > n$
2. **(Pseudoaleatoriedade.)** Para todo algoritmo probabilístico polinomial D, existe uma função negligenciável $\mathrm{negl}(n)$ tal que

$$\left|\Pr\left[D\left(G(s)\right) = 1\right] - \Pr[D(r) = 1]\right| \leq \mathrm{negl}(n)$$

Onde a primeira probabilidade é sobre $s \leftarrow \{0,1\}^n$ e a aleatoriedade de D, e a segunda sobre $r \leftarrow \{0,1\}^{\ell(n)}$ e a aleatoriedade de D

Visualizando o “distinguidor” D



- D = algoritmo probabilístico polinomial que recebe uma string de tamanho $\ell(n)$
  - Aleatória ou pseudoaleatória
- D é um bom distinguidor se a sua probabilidade de aceitação muda de maneira perceptível em relação às duas possibilidades acima

Exemplo de PRG não seguro

- Defina $G(s) = s || \bigoplus_{i=1}^{n} s_i$, com expansão $\ell(n) = n + 1$
- Por que G não é um PRG?
- Distinguidor (*Distinguisher*) D:
  - Ao receber $y \in \{0,1\}^{n+1}$ como entrada, responda 1 se e somente se o último bit de y é igual ao XOR dos outros bits
  - Se y é pseudoaleatório $\big(y = G(s)\big)$, qual a probabilidade de que $G(y) = 1$?
    - Exatamente 1
  - Se y é aleatório, qual essa probabilidade?
    - Exatamente $1/2$
  - Como $|1 - 1/2| = 1/2$, G não é um PRG

Experimento $\mathrm{PseudoR}_{\mathcal{A},G}(n)$

1^n

Desafiante



$s \leftarrow \{0,1\}^n$
$r \leftarrow \{0,1\}^\ell$
$b \leftarrow \{0,1\}$
Se b $= 0$, $y := r$
Se b $= 1$, $y := G(s)$

y

$b' =$ chute para o valor de b

Adversário $\mathcal{A}$
Tempo poli(n)



- G é um algoritmo que mapeia $\{0,1\}^n$ para $\{0,1\}^\ell$
- A saída do experimento $\mathrm{PseudoR}_{\mathcal{A},G}(n)$ é igual a 1 se $f(x') = y$, e 0 caso contrário

Equivalência do experimento

- **Teorema.** Seja ℓ um polinômio e G um algoritmo determinístico polinomial tal que para todo n e $s \in \{0,1\}^n$, $G(s)$ é uma string de tamanho $\ell(n)$. G é um gerador pseudoaleatório se e somente se

1. **(Expansão.)** Para todo n, vale que $\ell(n) > n$
2. **(Pseudoaleatoriedade.)** Para todo algoritmo probabilístico polinomial $\mathcal{A}$, existe uma função negligenciável $\mathrm{negl}(n)$ tal que

$$\Pr\left[\mathrm{PseudoR}_{\mathcal{A},G}(n) = 1\right] \leq \frac{1}{2} + \mathrm{negl}(n)$$

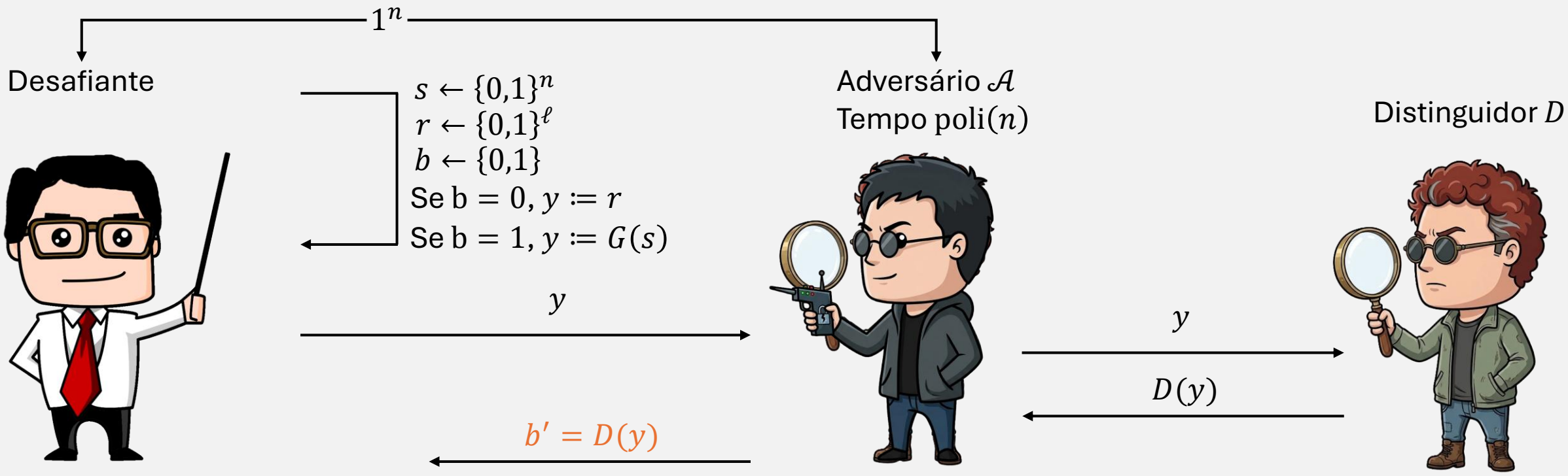
Demonstração da equivalência

- (experimento ⇒ Definição 3.14)
  - Considere um distinguidor D tal que (s.p.g.)

$$\Pr\left[D\left(G(s)\right) = 1\right] - \Pr[D(r) = 1] = \varepsilon > 0$$

  - Por que s.p.g.?
  - Como construir um adversário $\mathcal{A}$ para o experimento $\mathrm{PseudoR}_{\mathcal{A},G}(n)$ usando D?
  - Ao receber y, basta retornar exatamente $D(y)$
    - Afinal, D é “bom” justamente em detectar se uma string é pseudoaleatória ou aleatória

A redução



- Resposta de $\mathcal{A}$?
  - $D(y)$
  - $\mathcal{A} = D$

Demonstração da equivalência

- Lembrando que $\mathrm{PseudoR}_{\mathcal{A},G}(n) = 1$ exatamente quando $b' = b$

$$
\begin{aligned}
&\Pr\left[\mathrm{PseudoR}_{\mathcal{A},G}(n) = 1\right] \\
&= \Pr[\, b' = 1 \mid b = 1\,] \Pr[b = 1] + \Pr[\, b' = 0 \mid b = 0\,] \Pr[b = 0] \\
&= \frac{1}{2} \cdot \Pr\left[D\left(G(s)\right) = 1\right] + \frac{1}{2} \cdot \Pr[D(r) = 0] \\
&= \frac{1}{2} \cdot \Pr\left[D\left(G(s)\right) = 1\right] + \frac{1}{2}(1 - \Pr[D(r) = 1]) \\
&= \frac{1}{2} + \frac{1}{2} \cdot \left(\Pr\left[D\left(G(s)\right) = 1\right] - \Pr[D(r) = 1]\right) = \frac{1}{2} + \frac{\varepsilon}{2}
\end{aligned}
$$

- Pela segurança de acordo com o experimento:

$$\Pr\left[\mathrm{PseudoR}_{\mathcal{A},G}(n) = 1\right] \leq \frac{1}{2} + \mathrm{negl}(n)$$

- Segue que $\varepsilon \leq \mathrm{negl}(n)$
- Para a volta, basta fazer $D(y) \coloneqq \mathcal{A}(y)$, e o argumento é essencialmente o mesmo

Candidatos?

- Alguns candidatos (mais teóricos)
  - Gerador de Goldreich: avalia um predicado simples em vários subconjuntos da entrada
  - Blum-Blum-Shub: baseado no problema da Fatoração
  - Blum-Micalli: baseado no problema do Logaritmo Discreto
- Na prática
  - AES: uma cifra que usa a chave como semente e incrementa um contador para obter vários bits pseudoaleatórios a partir de permutações sucessivas
  - SHAKE128 / SHAKE256: Aplica diversas permutações na semente, baseadas em uma chave secreta
  - ChaCha20 / Salsa20: Aplicas diversos bit-shifts, somas e XORs em diversas partes da última string produzida, iniciando com uma chave secreta

PRGs e OWFs

- **Teorema.** Funções unidirecionais existem se e somente se geradores pseudoaleatórios existem
- De geradores pseudoaleatórios para funções unidirecionais
  - Relativamente direto
  - Exercício (opcional)
- De funções unidirecionais para geradores pseudoaleatórios
  - Muito mais difícil
    - Resultado de Hastäd, Impagliazzo, Levin e Luby, de 1999
  - Um caso específico é uma sugestão de projeto semestral, com *permutações* unidirecionais

Cifras de fluxo (*Stream Ciphers*)

- Uma variação de PRG que produz os bits desejado sob demanda
  - Mais utilizada na prática
- Consiste em um par de algoritmos $(\mathrm{Init}, \mathrm{GetBits})$ tal que
  - Init recebe uma semente s e um vetor IV de inicialização opcional, e tem como saída um estado inicial st_0
  - $\mathrm{GetBits}$ recebe um estado st_i, e tem como saída um bit y e um novo estado st_{i+1}

Cifras de fluxo e PRGs

- A partir de uma cifra de fluxo, podemos obter quantos bits quisermos
  - Em particular, para construir um PRG G_ℓ que mapeia n bits para $\ell(n)$ bits

ALGORITHM 3.16

Constructing G_ℓ from (Init, GetBits)

Input: Seed s and optional initialization vector IV

Output: $y_1, \ldots, y_\ell$

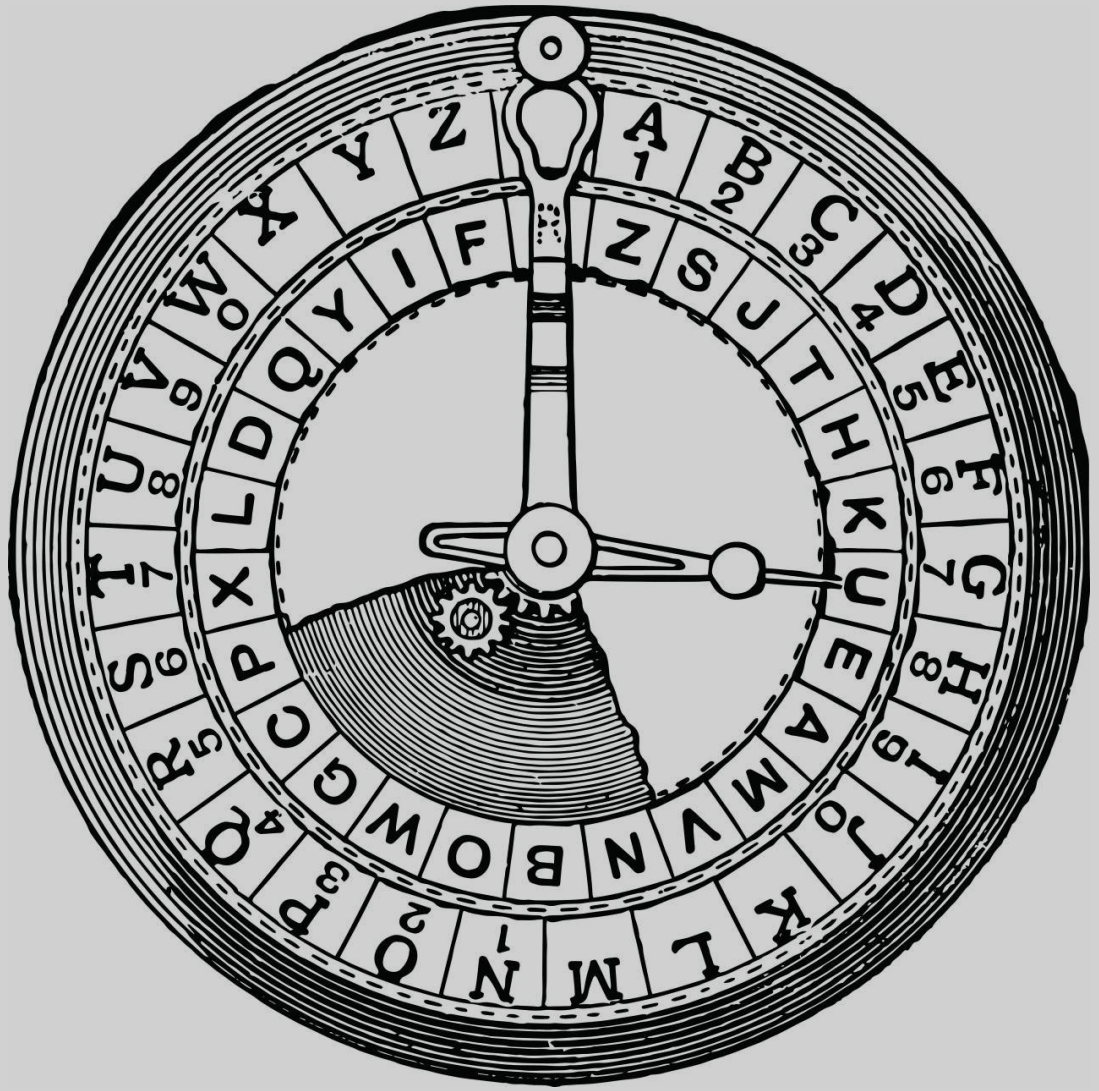
$\mathsf{st}_0 := \mathsf{Init}(s, IV)$

for $i = 1$ to ℓ:

  $(y_i, \mathsf{st}_i) := \mathsf{GetBits}(\mathsf{st}_{i-1})$

return $y_1, \ldots, y_\ell$

Para fechar



Para fechar

- Hoje
  - Primitivas Criptográficas
    - OWFs
    - PRGs
- Próxima aula
  - Construindo uma cifra computacionalmente segura com chave menor que a mensagem a partir de um PRG