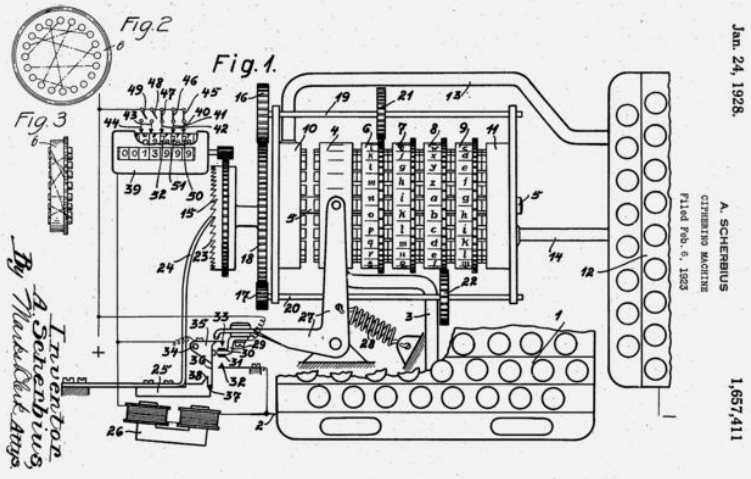
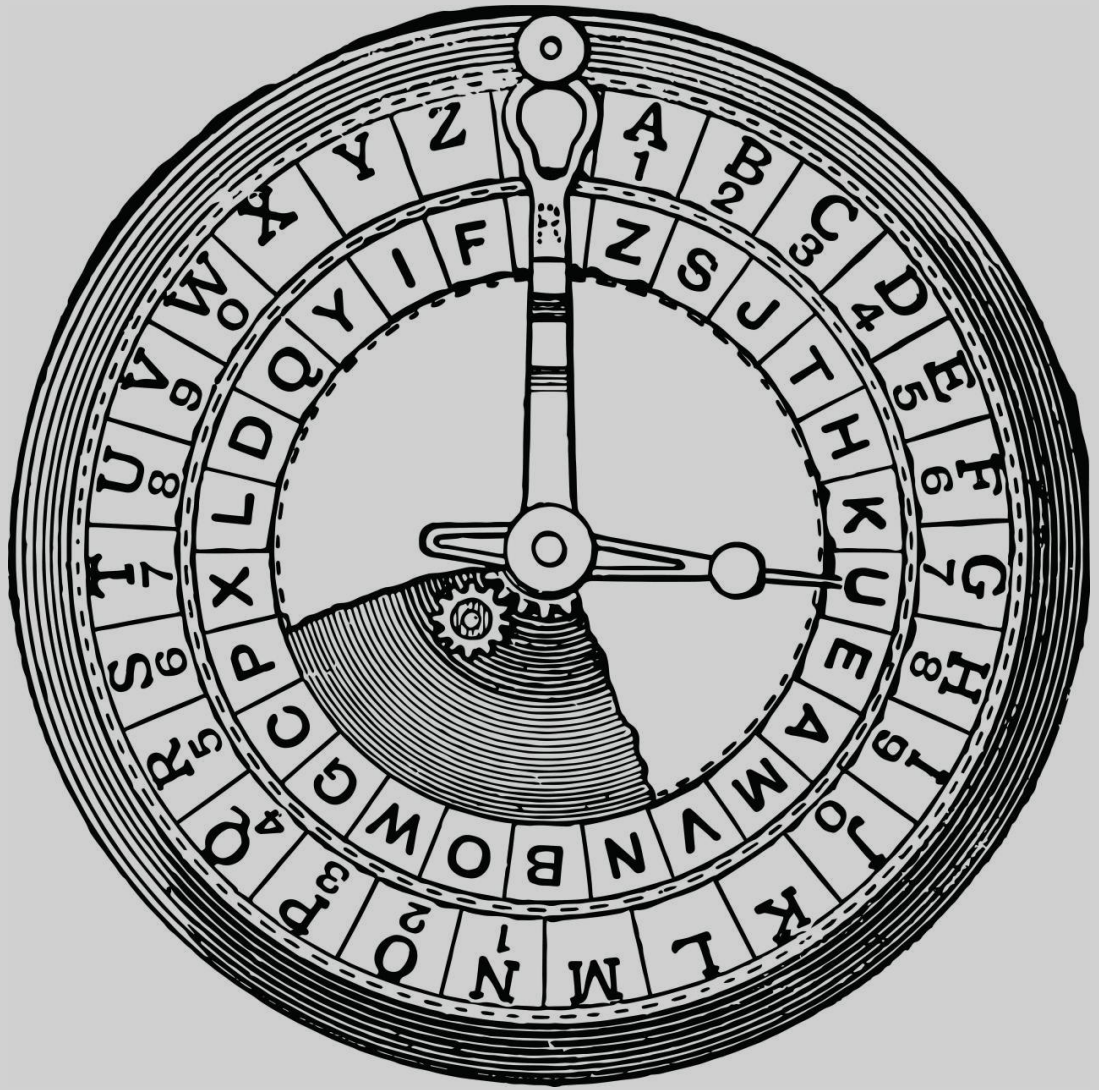
Funções Hash

Fundamentos da Criptografia

Nicollas Mocelin Sdroievski



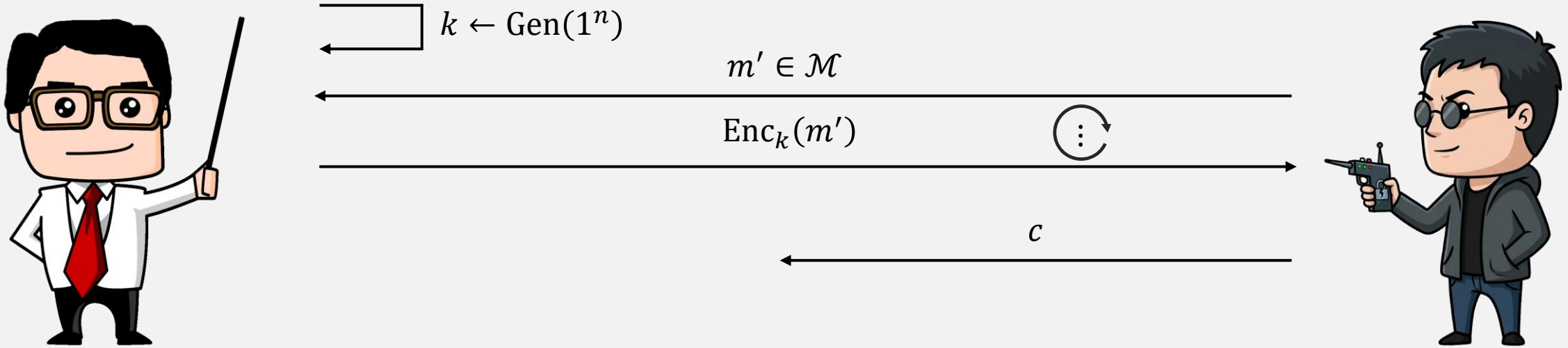
Na última aula



Experimento $\text{Enc-Forge}_{\mathcal{A},\Pi}(n)$

Desafiante

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



- Sejam $m := \text{Dec}_k(c)$ e Q o conjunto de todas as consultas que $\mathcal{A}$ fez para o oráculo de cifragem
- A saída do experimento é 1 se e somente se
  - $m \neq \perp$
  - $m \notin Q$

Definição

- **Definição 4.16.** Uma cifra simétrica Π é *infalsificável* se para todo adversário probabilístico polinomial $\mathcal{A}$, existe uma função negligenciável negl tal que:

$$\Pr\left[\text{Enc-Forge}_{\mathcal{A},\Pi}(n) = 1\right] \leq \mathrm{negl}(n)$$

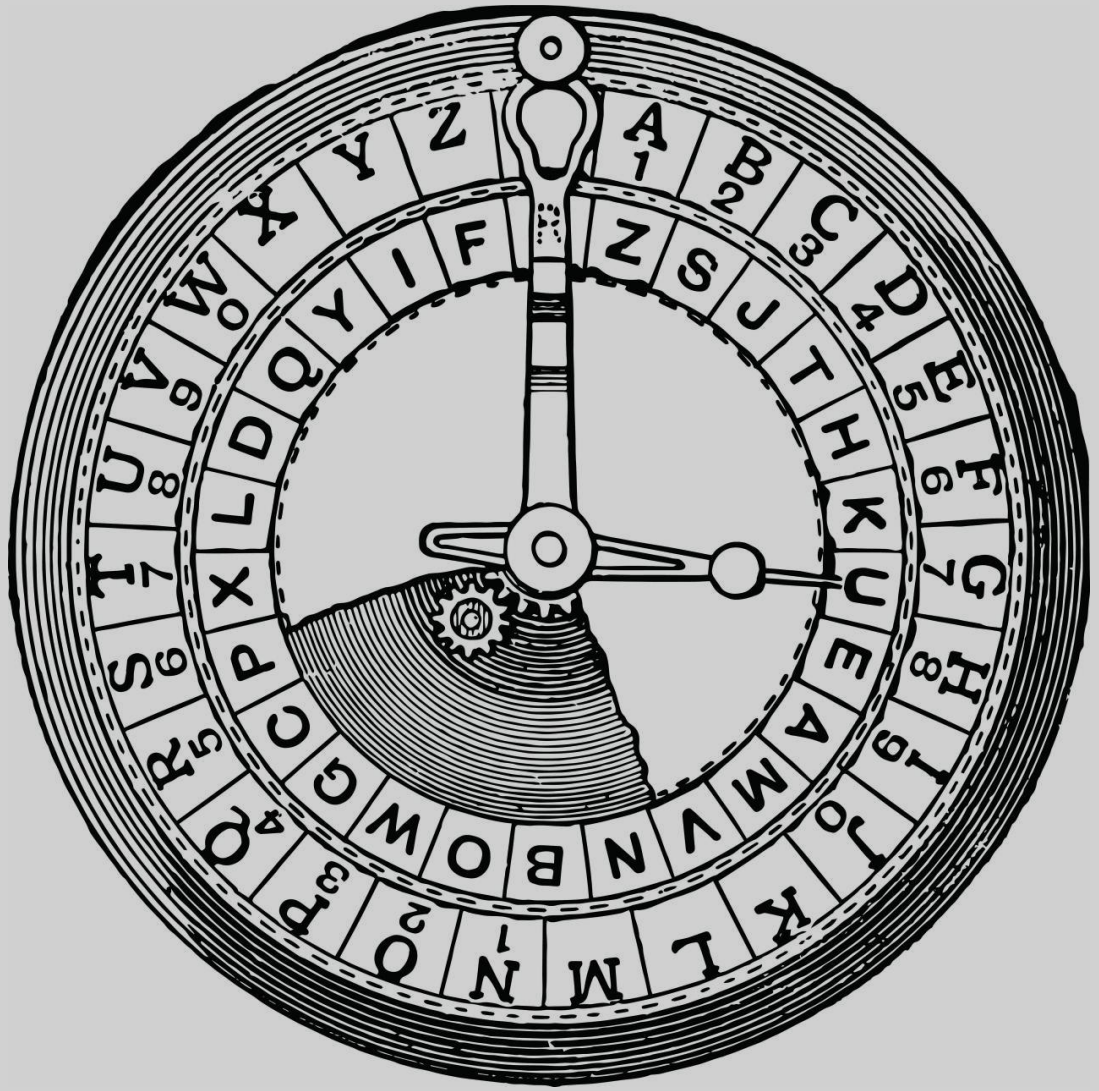
- **Definição 4.17.** Uma cifra simétrica é uma *cifra autenticada* se é CCA-segura e infalsificável.

Construções

- Combinar uma cifra segura com um MAC seguro
  - Sempre com chaves independentes!
- Nem todas as combinações funcionam

1. Cifrar-e-autenticar
   - Pode nem ser EAV-seguro
2. Autenticar-e-então-cifrar
   - Depende da construção
3. Cifrar-e-então-autenticar
   - Sempre seguro se a cifra é CPA-segura e o MAC fortemente seguro

Funções Hash



Ideia

- Uma função H que comprime a sua entrada, produzindo um *digest*
- Objetivo principal: evitar colisões
  - Colisão = duas entradas distintas x e y tal que $H(x) = H(y)$
  - Em estruturas de dados: garantias probabilísticas
    - A probabilidade de colisão (dada uma escolha aleatória de função hash) é "pequena"
    - Assume independência da função com as entradas x e y
  - Em Criptografia?
    - Um adversário eficiente só pode conseguir encontrar uma colisão com probabilidade negligenciável

Algumas aplicações

- Usar o valor de $H(x)$ como uma impressão digital de x
  - Dois arquivos são iguais se seus hashes são iguais
- Identificação de malware
  - Programas anti-malware guardam os hashes de vírus, Worms, trojans, ...
  - E comparam com os arquivos baixados ou escaneados
- Deduplicação
  - Bastante usado em serviços de armazenamento na nuvem
  - Se mais de um usuário deseja guardar o mesmo arquivo, mantém apenas uma cópia do arquivo
  - Como funciona?
    - O usuário quer guardar um arquivo x
    - O usuário envia $H(x)$ para o servidor
      - Se um arquivo com o mesmo hash existe, basta apontar para ele
      - Senão, faz upload do novo arquivo
- Compartilhamento de arquivos *peer-to-peer*
  - Usam tabelas de hashes para permitir buscas por arquivos

Definição

- Vamos considerar funções hash *chaveadas*
  - $H^s(x) \doteq H(s, x)$
  - Difícil de achar colisões dada uma chave s gerada de maneira aleatória
  - Comum que nem toda chave seja válida
    - Chave será gerada por um algoritmo Gen

- **Definição 5.1.** Uma *função hash* (com tamanho de saída ℓ) é um par (Gen, H) de algoritmos probabilísticos polinomiais que satisfazem as seguintes condições:
  - Gen é um algoritmo probabilístico que recebe um parâmetro de segurança 1^n e tem como saída uma chave s. Assumimos que n pode ser determinado a partir de s
  - H recebe uma chave s e uma string $x \in \{0,1\}^*$ e tem como saída $H^s(x) \in \{0,1\}^{\ell(n)}$

- A função H^s pode ser definida apenas para entradas de tamanho $\ell'(n) > \ell(n)$, e neste caso é uma *função hash de tamanho de entrada fixo* ou uma *função de compressão*

Compressão e segurança

- No caso de tamanho fixo, pedimos que $\ell'(n) > \ell(n)$
  - Por que?
    - Construção trivial com $\ell'(n) = \ell(n)$: função identidade
- No caso de tamanho de entrada arbitrário, a função hash comprime entradas suficientemente grandes
- Como definir a segurança?
  - Experimento

O experimento $\text{Hash-coll}_{\mathcal{A},H}(n)$

Desafiante

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



$s \leftarrow \text{Gen}(1^n)$

s

x, x'



- $\mathcal{A}$ obtém sucesso se e somente se $x \neq x'$ e $H^s(x) = H^s(x')$
- Neste caso escrevemos $\text{Hash-coll}_{\mathcal{A},H}(n) = 1$

Definição de segurança

- **Definição 5.2.** Uma função hash $\Pi = (\mathrm{Gen}, H)$ é *resistente à colisões* se para todos os adversários probabilísticos polinomiais $\mathcal{A}$ existe uma função negligenciável negl tal que

$$\Pr[\text{Hash-coll}_{\mathcal{A},H}(n) = 1] \leq \mathrm{negl}(n)$$

- Por simplicidade, vamos nos referir à H ou H^s como uma função hash resistente à colisões, quando tecnicamente só poderíamos dizer isso sobre (Gen, H)

Na prática, duas diferenças

- Tamanho de saída fixo ℓ
  - Assim como cifras de bloco na prática
- Não-chaveadas
  - Apenas uma função fixa $H : \{0,1\}^* \rightarrow \{0,1\}^\ell$
- Problemas teóricos com essas questões?
  - Existe algoritmo de tempo constante que acha uma colisão
  - Basta deixar uma colisão (x, x') “hardcoded” no algoritmo
  - Funções chaveadas resolvem esse problema
    - Impossível “hardcode” colisões para todas as chaves/parâmetros de segurança
- Para efeitos práticos, estas funções são resistentes a colisões
  - Colisões desconhecidas (e difíceis de encontrar)
  - Provas de segurança se estendem para esses caso desde que um adversário que “quebra” a segurança possa ser usado para ecnontrar uma colisão

Noções mais fracas de segurança

- **Resistência à segunda pré-imagem**
  - Dados s e uma escolha aleatória de x, é difícil encontrar $x' \neq x$ tal que $H^s(x') = H^s(x)$
- **Resistência à pré-imagem**
  - Dados s e uma escolha uniforme de y, é difícil encontrar x tal que $H^s(x) = y$

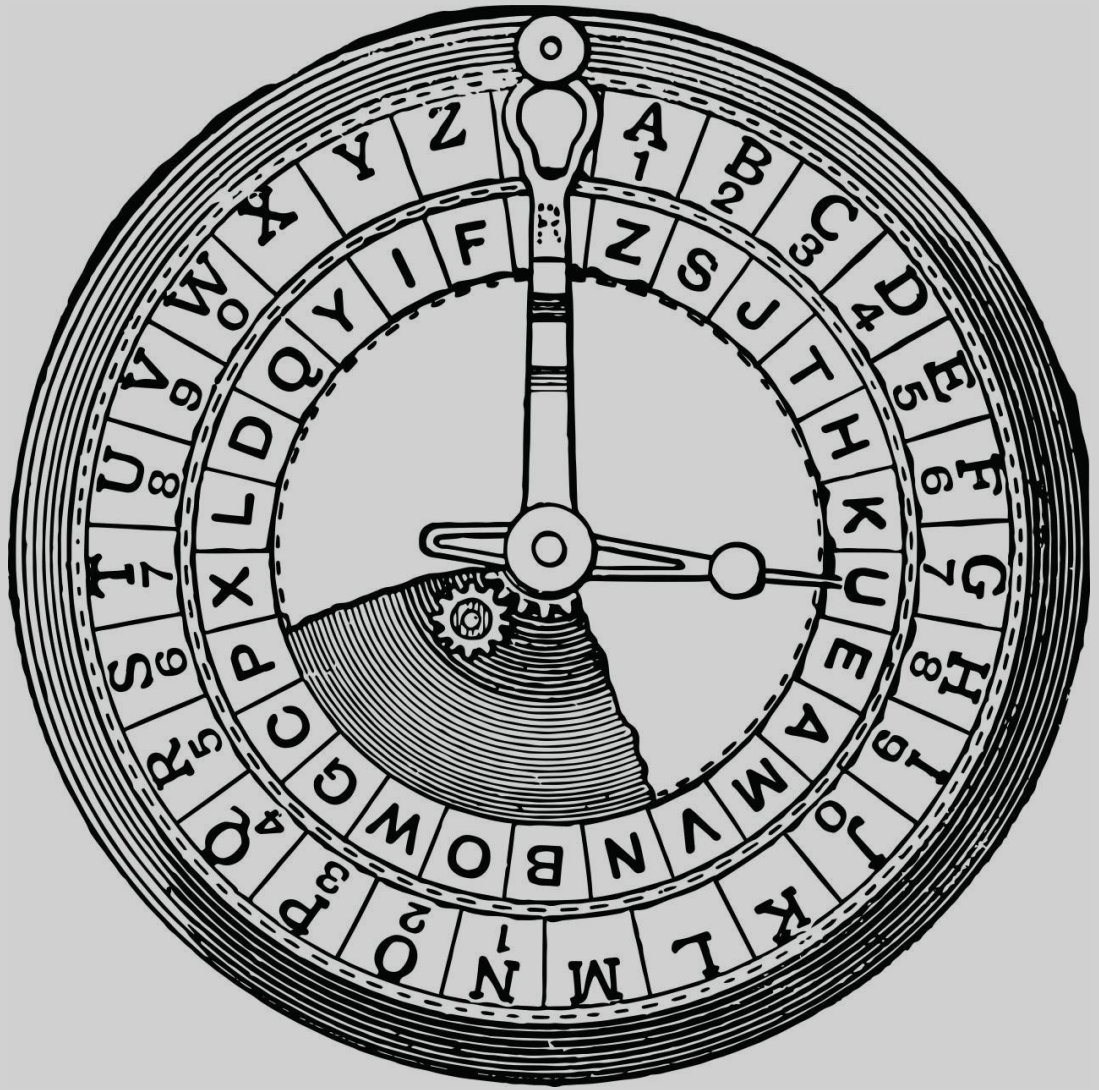
Colisões versus segunda pré-imagem

- Toda função hash que é resistente à colisões também é resistente à segunda pré-imagem
  - Por que?
  - Imagine que um adversário consegue, dada uma escolha aleatória de x, encontrar $x' \neq x$ tal que $H^s(x') = H^s(x)$
  - Então x e x' são uma colisão
- Como formalizar a redução?

Segunda versus primeira pré-imagem

- Toda função hash que é resistente à segunda pré-imagem também é resistente à pré-imagem
  - Por que?
  - Imagine que um adversário consegue, dada uma escolha aleatória de y, encontrar x tal que $H^s(x) = y$
  - Basta sortear x', computar $y = H^s(x')$ e então encontrar x tal que $H^s(x) = y$
  - Com alta probabilidade, $x' \neq x$ e encontramos uma segunda pré-imagem
- Como formalizar a redução?

A transformação de Merkle-Damgård

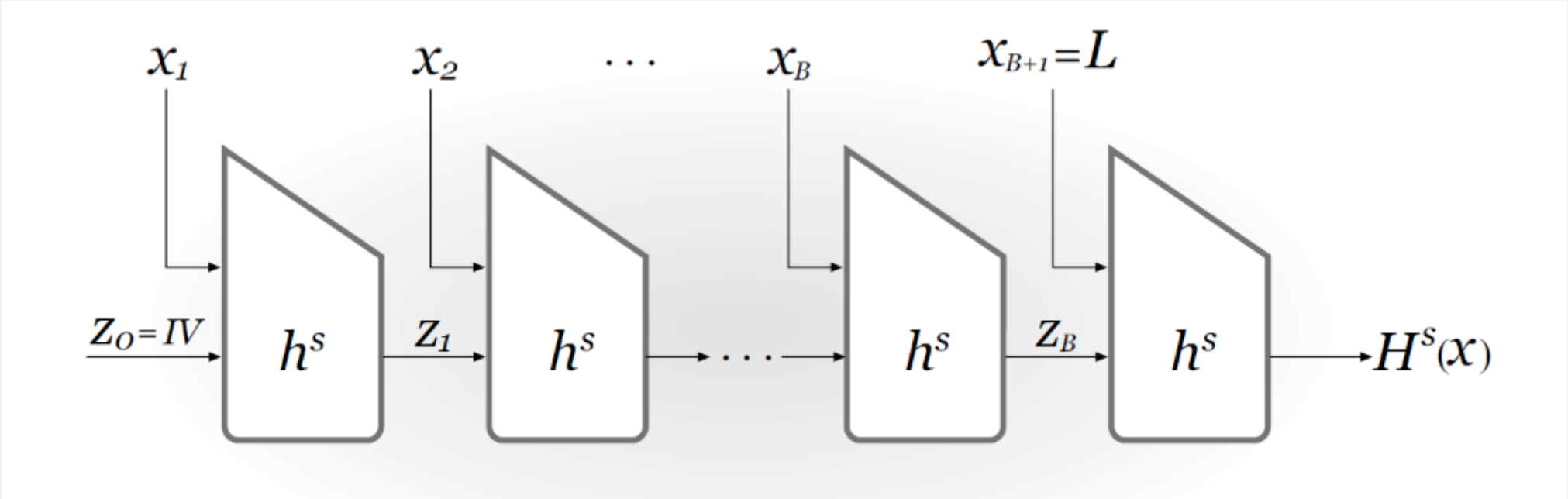


Extensão de domínio

- Construções mais comuns de funções hash
  1. Construir uma função hash para tamanho de entrada fixo
  2. Usar extensão de domínio para que a função se aplique à entradas de tamanho arbitrário
- A transformação de Merkle-Damgård permite essa extensão
- Muito usada na prática (MD5, família SHA)
- Importância teórica: comprimir por um bit é tão fácil (ou difícil) quanto comprimir arbitrariamente

Ideia

- Imaginar que temos uma função que mapeia $2n$ bits para n bits
  - A construção funciona para qualquer fator de compressão
- Para comprimir $x_1, \ldots, x_{B+1}$
- Computar uma sequência de hashes de $x_i || z_{i-1}$, onde z_{i-1} é o último hash computado
  - O último hash é a saída



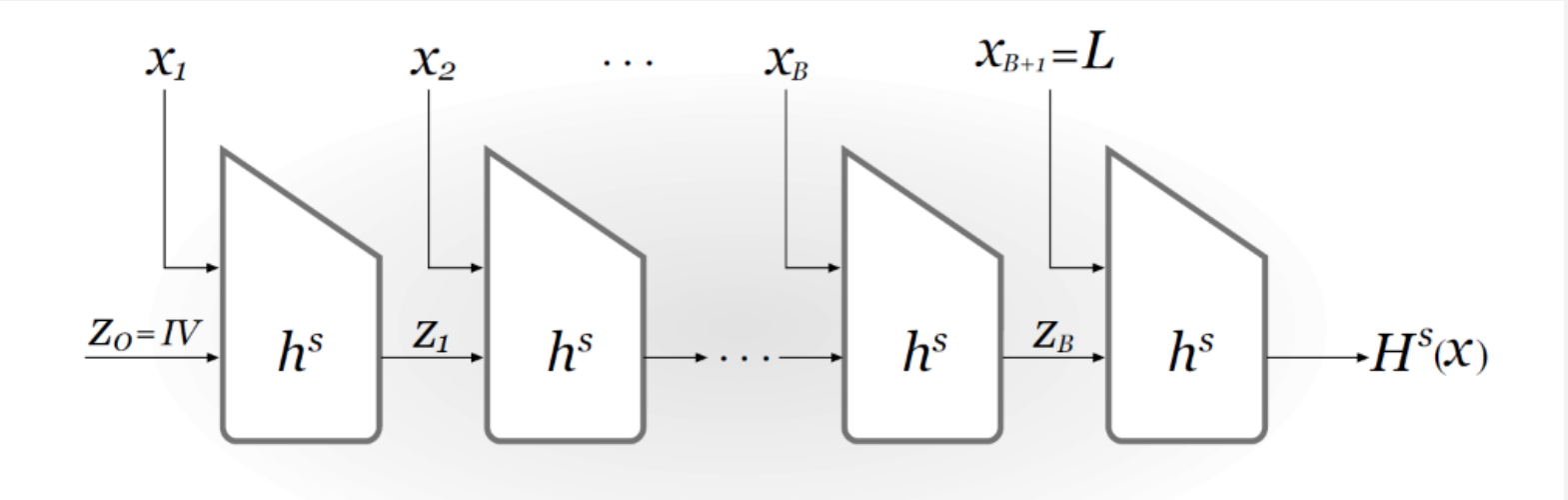
Construção 5.3

- Seja (Gen, h) uma função hash para entradas de tamanho fixo que mapeia entradas de tamanho $2n$ para entradas de tamanho n
- Construímos (Gen, H) a seguir:
  - Gen é o mesmo
  - H: dados s e $x \in \{0,1\}^*$ de tamanho $L < 2^n$, faça o seguinte:
    1. Faça $B = \left\lceil \frac{L}{n} \right\rceil$ (o número de blocos em x). Preencha x com zeros para que tenha tamanho múltiplo de n, obtendo $x_1, \ldots, x_B$. Faça $x_{B+1} = L$
    2. Faça $z_0 = 0^n$
    3. Para $i = 1, \ldots, B + 1$, compute $z_i = h^s(z_{i-1} || x_i)$
    4. Retorne z_{B+1}

Segurança

- **Teorema 5.4.** Se (Gen, h) é resistente à colisões, então (Gen, H) também é.
- **Demonstração:** Mostramos que uma colisão em H^s nos permite encontrar uma colisão em h^s
- Sejam $x \neq x'$ duas strings de tamanho L e L', respectivamente, tal que $H^s(x) = H^s(x')$
  - Sejam ainda $x_1, \ldots, x_B$ os blocos preenchidos de x, e $x'_1, \ldots, x'_{B'}$ os blocos preenchidos de x'

Continuação da demonstração

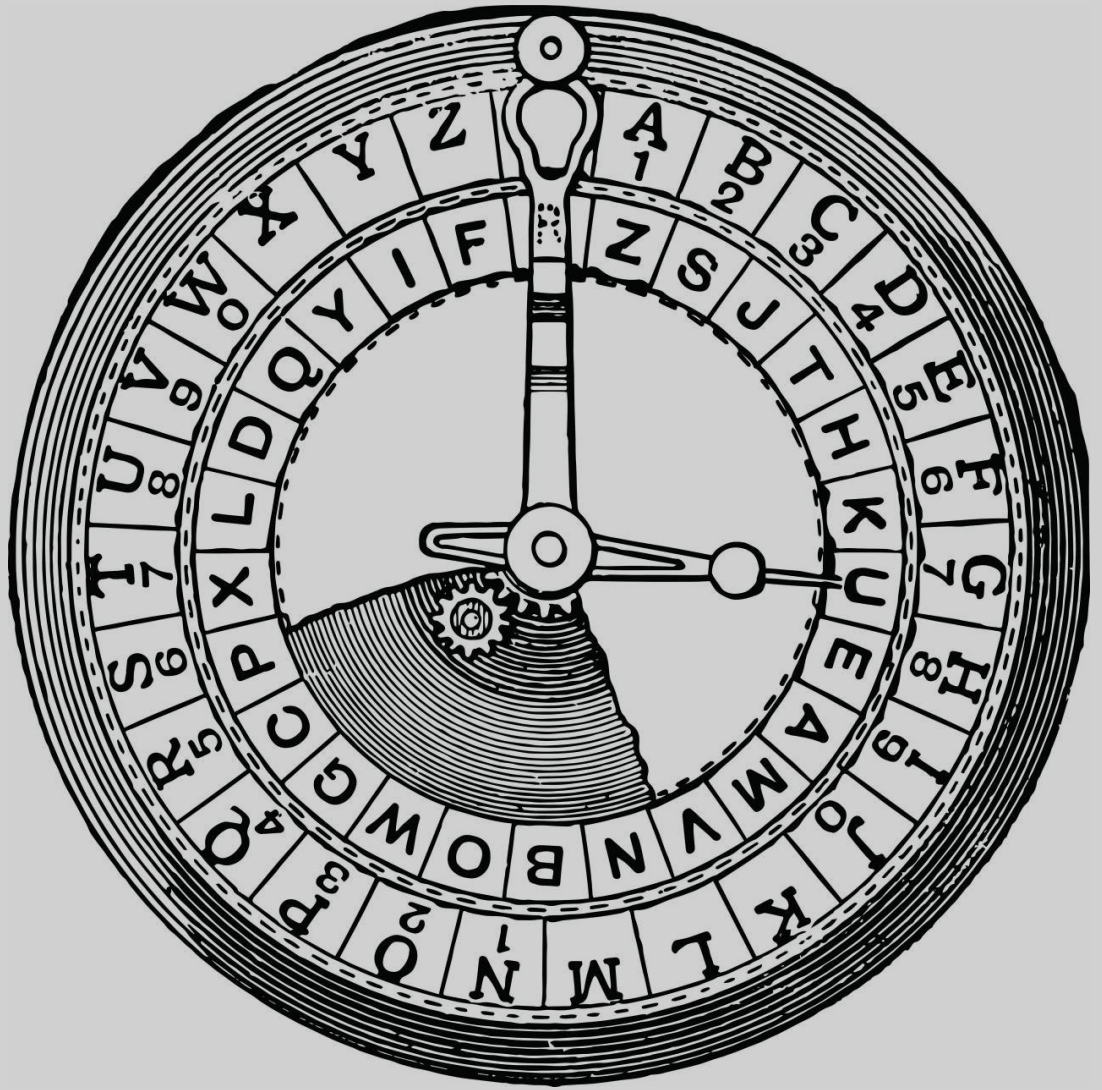


- Consideramos dois casos:
- Caso 1: $L \neq L'$
  - Considere $H^s(x) = z_{B+1} = h^s(z_B||L)$ e $H^s(x') = z'_{B'+1} = h^s(z'_{B'}||L')$
- Como $H^s(x) = H^s(x')$, temos que $h^s(z_B||L) = h^s(z'_{B'}||L')$, mas como $L \neq L'$, essa é uma colisão em h^s

Continuação da demonstração

- Caso 2: $L = L'$
  - Agora $B = B'$ e $L = L'$, então a estratégia de antes não funciona
  - Seja $z_0, \ldots, z_{B+1}$ a sequência de valores definida na computação de $H^s(x)$
  - Defina $I_i \doteq z_{i-1} || x_i$ como a i-ésima entrada para h^s e $I_{B+2} = z_{B+1}$
  - Defina I'_i e I'_{B+2} de maneira análoga para x'
  - Seja N o maior índice tal que $I_N \neq I'_N$, que deve existir (por que?)
  - Como
$$I_{B+2} = z_{B+1} = H^s(x) = H^s(x') = z'_{B+1} = I'_{B+2}$$
  - Temos $N \leq B + 1$
- Como $I_{N+1} = I'_{N+1}$ e $z_N = z'_N$ em particular, I_N e I'_N formam uma colisão em h^s

Um ataque genérico à funções hash



Nosso modelo de análise

- Fixamos $H : \{0,1\}^* \to \{0,1\}^\ell$
- Ataque trivial?
  - Força bruta
  - Tempo $O(2^\ell)$
- Assumimos que o valor de $H(x)$ é uniforme
  - Essencialmente o pior caso
  - Mais fácil encontrar colisões quando H não é “tão aleatória”

Ataque do aniversário

- Escolhemos q valores $x_1, \dots, x_q$ distintos para tentar encontrar uma colisão
- Lembre que o valor de $H(x)$ é aleatório uniforme
- Qual a probabilidade de que uma colisão ocorra?
  - A análise é equivalente à análise do problema do aniversário
  - Se q pessoas estão em uma sala, qual a probabilidade de que duas tenham o mesmo aniversário (com cada data escolhida dentre 365 possíveis)?
    - Com 23 pessoas, aproximadamente $1/2$
  - No nosso caso
    - Sorteamos q imagens $y_1, \dots, y_q$ em $\{0,1\}^{\ell}$
    - E queremos saber a probabilidade de que existam $i \neq j$ tal que $y_i = y_j$

Análise

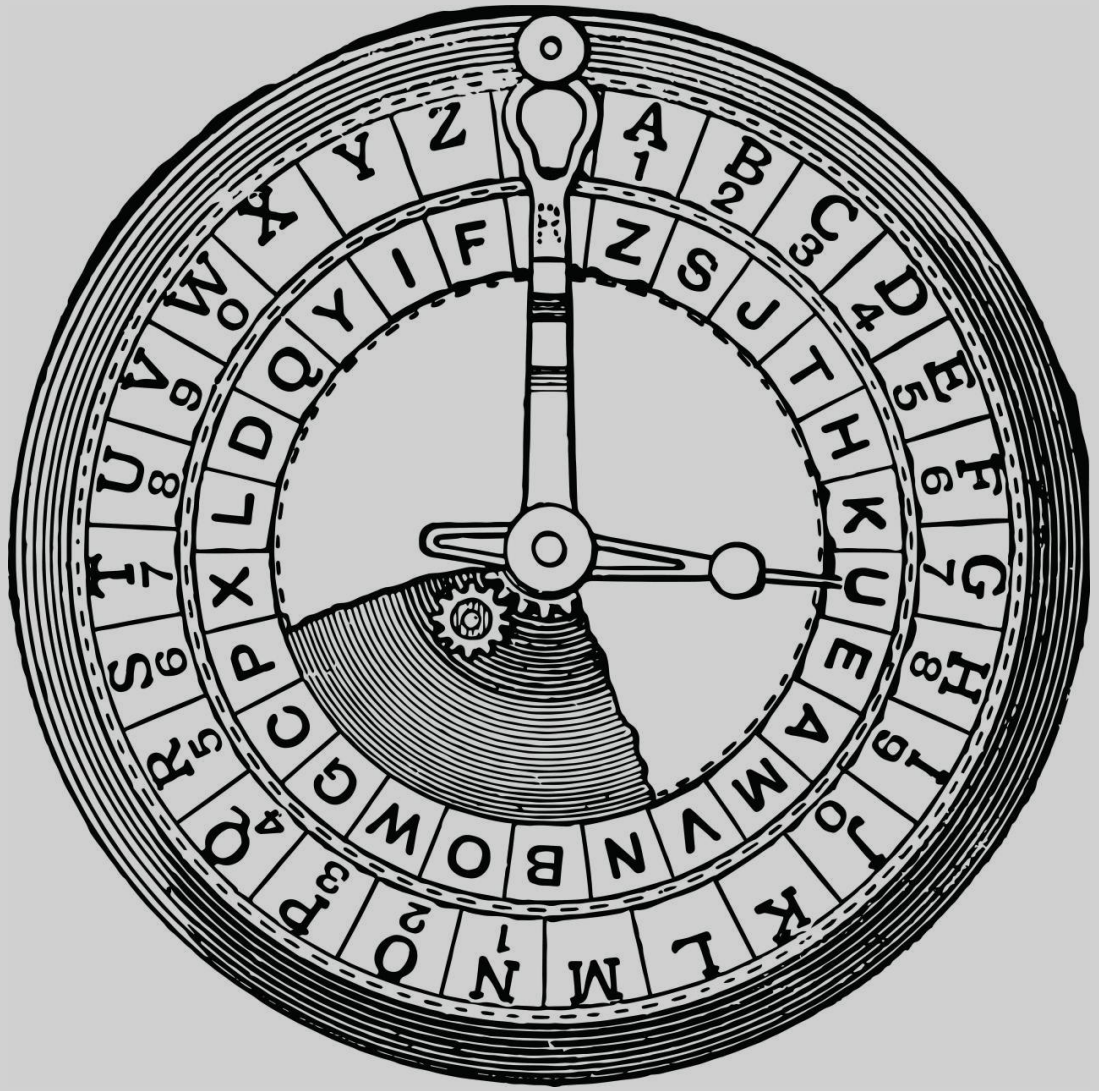
- **Lema A.16.** Fixe um inteiro positivo N e considere escolher $q \leq \sqrt{2N}$ elementos $y_1, \ldots, y_q$ de maneira aleatória e uniforme de um conjunto de tamanho N. Então a probabilidade de que existam $i \neq j$ tal que $y_i = y_j$ é de pelos menos $\frac{q(q-1)}{4N}$.

- Usando o Lema, basta escolher $q = \Theta\left(2^{\ell/2}\right)$ para garantir probabilidade de colisão aproximadamente $1/2$
- Se queremos que nossa função hash seja resistente à colisões contra adversários de tempo T, precisamos que $\ell \geq 2 \log T$

Encontrando colisões mais interessantes

- No geral, queremos encontrar colisões entre duas mensagens x_1 e x_2 que "façam sentido"
- Lembramos que o ataque só pede que as entradas sejam distintas, não aleatórias!
- Basta então, gerar $q = \Theta(2^{\ell/2})$ mensagens "similares" à x_1 e $q = \Theta(2^{\ell/2})$ mensagens "similares" à x_2
  - Uma análise semelhante ao Lema A.16 garante uma probabilidade próxima de $1/2$ de sucesso
- **Observação:** uma mensagem com 64 palavras que admitem um sinônimo já permite 2^{64} variações "similares"

Para fechar



Para fechar

- Hoje
  - Funções hash criptográficas
    - Comprimem a entrada
    - Inviável encontrar colisões
  - Merkle-Damgård
  - Alguns ataques genéricos
- Próxima aula
  - Funções Hash e MACs
  - O modelo de oráculo aleatório (Random-Oracle model)