Funções resumo (*Hash functions)*

Prof. Avelino Francisco Zorzo
Escola Politécnica - PUCRS

1

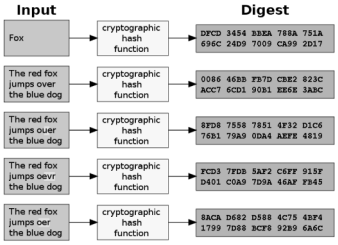
Funções resumo (*Hash*)

- Comprime uma mensagem de tamanho arbitrário em uma mensagem de tamanho fixo, por exemplo, *checksum*.
  - Usado desde 1950s
  - Facilita a detecção de erros ou comparação de conteúdo

2

Funções resumo criptográficas

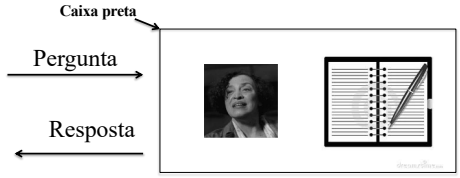
- Inventadas para assinatura digital
  - Provê garantia de integridade de dados
  - Efeito avalanche



(Wikipedia)

3

Abstração: oráculo aleatório



If pergunta nova

Oráculo devolve um string de tamanho fixo e marca o registo no seu livro

Else

Ela procura no livro e devolve a mesma resposta.

4

Mas um oráculo aleatório ideal é impossível

- Como garantir que cada saída represente somente uma entrada?

- Isto é teoricamente impossível
  - O tamanho do espaço de mensagens é muito maior que o tamanho do espaço de respostas
- Solução prática
  - Garantir que seja computacionalmente difícil encontrar duas mensagens que gerem a mesma saída

5

Requisito de segurança

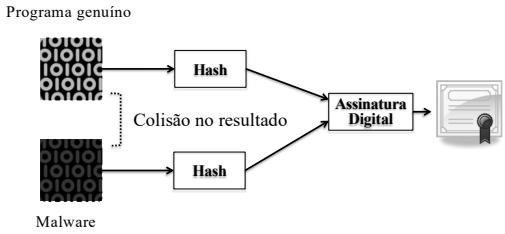
- Resistência de pré-imagem
  - DadoH(m), não é possível encontrar m
- Segunda resistência de pré-imagem
  - DadoH(m_1), não acha uma mensagem m_2, de forma que H(m_1) = H(m_2)
- **Resistente a colisão**
  - Não pode encontrar duas mensagens m_1 e m_2, de forma que H(m_1) = H(m_2)

6

Famílias de funções resumo

- MD5 (1991) – (Message-Digest Algorithm 5)
  - 128-bit message digest
  - Quebrado por Wang Xiaoyun *et alli* em 2005
- NIST: Secure Hash Algorithm
- SHA-1 (1995) – Secure Hash Algorithm
  - 160-bit message digest
  - Inseguro (2^{69}, Wang Xiaoyun *et alli* 2005)
- SHA-2 (2001)
  - SHA-256
  - SHA-512

Colisão é perigoso



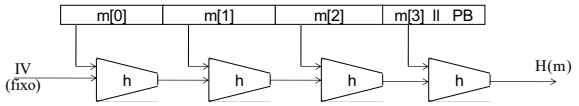
Exemplos no mundo real: Flame malware

- Detectado pelo Irã CERT em Maio de 2012
- Malware para espionagem
- Explorava colisão no MD5
  - Microsoft Terminal Server Licensing Service certificate ainda usava MD5
  - Produzia uma assinatura digital falsificada que parecia ter sido originada da Microsoft

Princípio de projeto de funções resumo

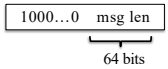
- Uma função resumo típica envolve 3 componentes no seu projeto
  - Modo de operação
  - Uma estrutura de compressão
  - Operações de confusão e difusão

Merkle-Damgard



[Introduction to Cryptography – Dan Boneh]

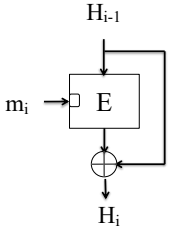
- PB: bloco padding



- Teorema: Se a função de compressão é resistente a colisão, então a função resumo também é

Funções de compressão

Davies-Meyer (used in MD5, SHA-1, SHA-2)

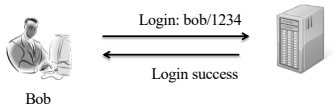


- E é uma cifra de bloco
- Use a mensagem como chave
- $h(H,m)=E(m,H) \oplus H$
- Compressão
  - Tamanho da entrada: tamanho da chave + tamanho do bloco
  - Tamanho da saída: tamanho do bloco

Aplicações de funções resumo

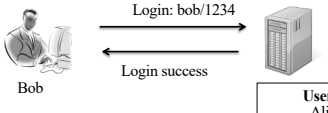
- Assinatura Digital
- Integridade de Dados
  - Exemplo: Checksum baixar um software
- Gerador de números aleatórios
- Privacidade de Dados
  - Proteger senhas em texto claro

Autenticação de senhas



Compara a senha em uma base de dados

Username: senha
Alice: 19890102
Bob: 1234
Charlie: jesmond



Calcula a função resumo da senha e compara com o que está na base de dados

Username: H(pw, salt), Salt
Alice: 8ced834745…, 8FA
Bob: ff4ed0bd13d2…, E9A
Charlie: 32cbba4a1…, 48C

Triste: problemas no mundo real

- 2012-09-25: IEEE sofreu uma invasão
  - 100,000 Plaintext passwords vazaram.
- 2012-07-12: Base do Yahoo Voices invadido
  - 0,5 milhões de plaintext passwords vazaram
- 2012-06-07: Banco de dados do LinkedIn invadido.
  - 6.5 milhões de Unsalted hashes disponibilizadas online.
- 2010-12-22: Contas do Gawker Media invadidas
  - 1.3 milhões de plaintext passwords vazadas

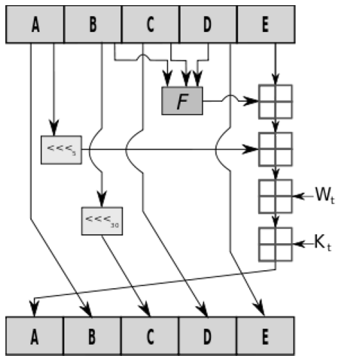
Importante

- Usar função resumo com Salt em senhas é uma boa prática
  - Torna a vida do atacante mais difícil para recuperar as senhas, mas não impossível
- Ataque de dicionário
  - Dado H(pw, salt) e salt
  - Um atacante pode tentar todas as senhas
  - O ataque é possível pois senhas tem baixa entropia

SHA1

Prof. Avelino Francisco Zorzo
Escola Politécnica - PUCRS

SHA-1



Source: Wikipedia

SHA-1

- *Initialize variables:*
  - h0 = 0x67452301
  - h1 = 0xEFCDAB89
  - h2 = 0x98BADCFE
  - h3 = 0x10325476
  - h4 = 0xC3D2E1F0
- *Pre-processing:* append the bit '1' to the message append $0 \le k < 512$ bits '0', so that the resulting message length (in *bits*) is congruent to 448 (mod 512) append length of message (before pre-processing), in *bits*, as 64-bit integer

19

SHA-1

- Process the message in successive 512-bit chunks
  - Break chunk into sixteen 32-bit big-endian words
  - Extend the sixteen 32-bit words into eighty 32-bit words
  - Initialize hash value for the chunk
  - Main loop
  - Add chunk's hash to result
- Produce the final hash value
  - digest = hash = h0 || h1 || h2 || h3 || h4

20

20

SHA-1

- Process the message in successive 512-bit chunks:
  - Break message into 512-bit chunks
  - **for** each chunk break chunk into sixteen 32-bit words w[i], $0 \le i \le 15$
  - Extend the sixteen 32-bit words into eighty 32-bit words:

```
for i from 16 to 79
  w[i]=(w[i-3] xor w[i-8] xor w[i-14] xor w[i-16]) leftrotate 1
```

- Initialize hash value for the chunk:
  - a = h0; b = h1; c = h2; d = h3; e = h4;

21

21

SHA-1

- Main loop
  - **for** i **from** 0 **to** 79
    - » **if** $0 \le i \le 19$ **then**
      - f = (b **and** c) **or** ((**not** b) **and** d); k = 0x5A827999
    - » **else if** $20 \le i \le 39$
      - f = b **xor** c **xor** d; k = 0x6ED9EBA1
    - » **else if** $40 \le i \le 59$
      - f = (b **and** c) **or** (b **and** d) **or** (c **and** d); k = 0x8F1BBCDC
    - » **else if** $60 \le i \le 79$
      - f = b **xor** c **xor** d; k = 0xCA62C1D6
    - » temp = (a **leftrotate** 5) + f + e + k + w[i]
    - » e = d; d = c; c = b **leftrotate** 30; b = a; a = temp

22

22

SHA-1

- Add chunk's hash to result so far
  - h0 = h0 + a
  - h1 = h1 + b
  - h2 = h2 + c
  - h3 = h3 + d
  - h4 = h4 + e

23

23