PMR3412 - Redes Industriais - 2021

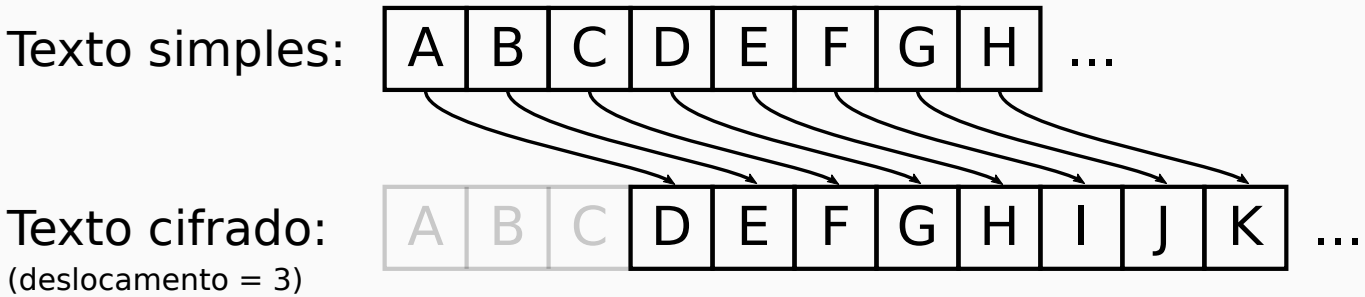
Aula 8 - Segurança: Conceitos Básicos - Hashing

Prof. Dr. André Kubagawa Sato
Prof. Dr. Marcos de Sales Guerra Tsuzuki

7 de Outubro de 2021

PMR-EPUSP

- Método rudimentar de criptografia utilizado por Júlio César (aprox. 100 a.C.).
- Cada letra é substituída por outra, deslocado de X posições no alfabeto.
- Exemplo com deslocamento igual a 3:
  - Texto simples: a ligeira raposa marrom saltou sobre o cachorro cansado
  - Texto cifrado: D OLJHLUD UDSRVD PDUURP VDOWRX VREUH R FDFKRUUR FDQVDGR
- Fácil de ser decifrado. Conseguem pensar em algumas estratégias?
- Memorize o princípio de Kerckhoff: **um sistema criptográfico deve ser seguros mesmo se tudo é conhecido sobre ele, exceto a chave**.



Criptografia - Usos

- No mundo interconectado de hoje, a criptografia está em toda parte.
- A quantidade e velocidade de troca de informações na Internet é impressionante e, incrivelmente, a maior parte deveria estar protegida de alguma forma.
- Isto é, apesar de existirem aproximadamente 4 bilhões de usuários na Internet, praticamente toda informação transmitida é direcionada para apenas uma pequena porcentagem destes.
- A criptografia é a principal ferramenta para providenciar proteção para informação. Ela fornece as seguintes proteções:

Criptografia - O que pode dar errado?

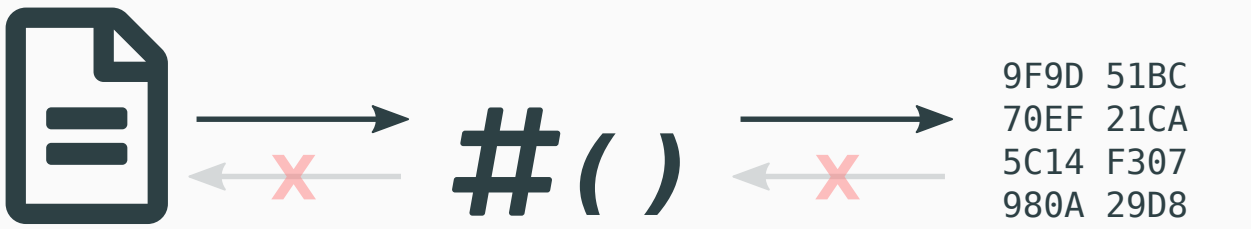
- Existem inúmeras formas para a criptografia dar errado; na realidade, existem muitas mais possibilidades de uso incorreto do que o contrário.
- Dois importantes motivos para este fato são:
  1. Criptografia é baseada em matemática bastante esotérica que a maioria dos desenvolvedores, engenheiros e profissionais de TI têm pouca experiência
  2. O uso correto da criptografia depende do contexto (reflita no caso da Cifra de César nos dias de hoje). Na prática, isto significa que boa parte do aprendizado de criptografia recai em como os vários parâmetros de configuração impactam a operação.
- Sendo assim, neste curso, apesar de desenvolvermos algumas aplicações de criptografia, esteja sempre ciente que isto é apenas para efeito didático. Lembre-se sempre:

YANAC: You Are Not A Cryptographer
(Você não é um Criptógrafo)

Hashing

Hashing - Introdução

- Hashing é um dos pilares de segurança criptográfica; envolve o conceito de *one-way function* ou *fingerprint*.
- Funções de hash funcionam bem quando seguem duas regras:
  1. elas produzem valores únicos e repetíveis para cada entrada; e
  2. o valor de saída não sugere nenhuma pista sobre a entrada que o produziu.
- Exemplos de hash functions: SHA-256 (boa), MD5 e SHA-1 (não tão boas).



Texto Simples

Função Hash

Hash, fingerprint ou digest

- Tamanho fixo
- Sempre produz o mesmo resultado
- Parece "randômico"

Hashing - Biblioteca `hashlib` do Python

- O Python possui a biblioteca hashlib (`https://docs.python.org/3/library/hashlib.html`), que disponibiliza diversas funções hash.
- Atenção: por simplicidade, estamos utilizando a função MD5 neste exemplo, que não é considerada segura!

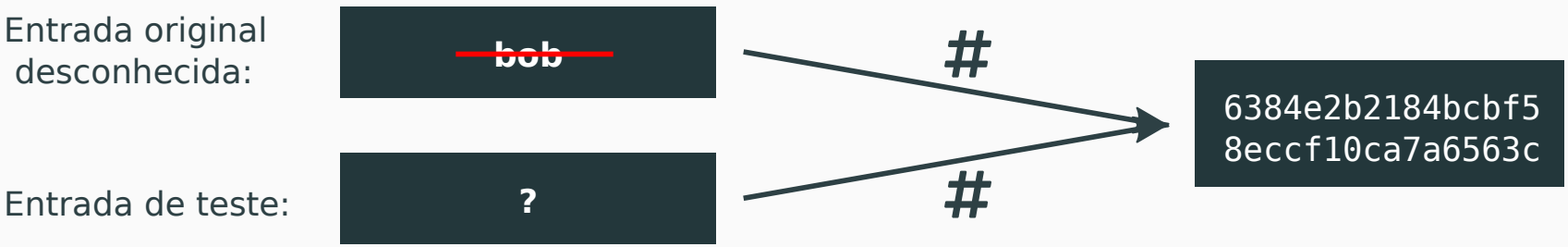
```
import hashlib
hashlib.md5(b'alice').hexdigest()
# saída: '6384e2b2184bcbf58eccf10ca7a6563c'
hashlib.md5(b'bob').hexdigest()
# saída: '9f9d51bc70ef21ca5c14f307980a29d8'
```

- O que ocorre se buscarmos no Google os seguintes digests? (clique nos digests para buscar)
  1. `5f4dcc3b5aa765d61d8327deb882cf99`
  2. `d41d8cd98f00b204e9800998ecf8427e`
  3. `6384e2b2184bcbf58eccf10ca7a6563c`

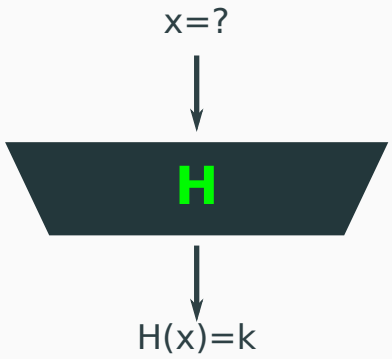
- Funções hashes podem ser criptográficas ou não – um exemplo de função hash não criptográfica é a paridade.
- Essencialmente, as funções hash mapeiam um número enorme (ou infinito) de coisas em um conjunto relativamente pequeno – definindo uma operação *lossy*.
- Sendo assim, todas as funções hash, incluindo as não criptográficas, possuem três qualidades fundamentais: **consistência**, **compressão** e **lossiness**.
- Além destas, para uma hash ser criptográfica, ou segura, deve ter as seguintes propriedades:

Hashing - Preimage resistance

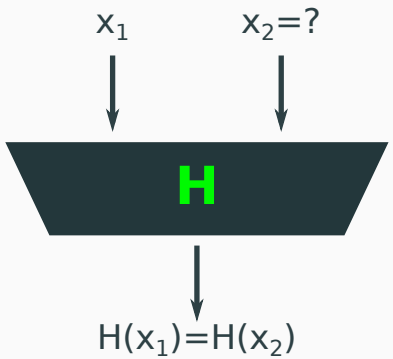
- A *preimage* pode ser definida como o conjunto de entradas para uma função hash que *produz um saída específica*, isto é
- **Preimage**: Uma *preimage* para um função hash H e uma valor hash k é o conjunto de valores de x tal qual $H(x) = k$.
- **Preimage resistance**: dado um *digest* de origem desconhecida, não é possível achar sequer um elemento da *preimage* sem ter que realizar uma quantidade irreal de trabalho.
- O processo de obter um elemento da *preimage* a partir da saída é chamado de inverter o hash. Assim, *preimage resistance* significa que encontrar o inverso é difícil. Em geral, para tentar obter a inversa é utilizado um ataque de força bruta, testando a função hash com inúmeras entradas diferentes.



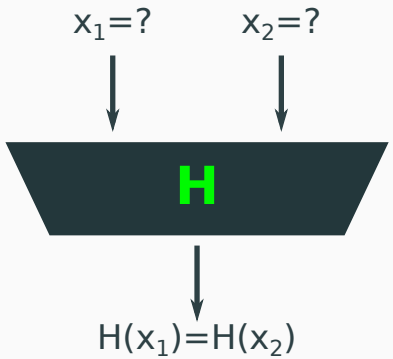
- **Second-Preimage Resistance**: se você já possui um documento que produz um determinado *digest*, é difícil encontrar um documento diferente que produz o mesmo *digest*. Ou, se você já possui o primeiro membro da *preimage*, não é mais fácil encontrar um segundo membro da mesma *preimage*
- **Collision Resistance**: significa que é difícil encontrar duas entradas quaisquer que produzem a mesma saída. Ou seja, é difícil gerar propositalmente duas entradas com um mesmo *digest*.



Preimage Resistance



Second-Preimage Resistance



Collision Resistance

Hashing - A propriedade Avalanche

- Uma propriedade que contribui para a *colision resistance* é **propriedade avalanche**: uma mudança na entrada, não importa quão pequena, causa uma grande e imprevisível mudança na saída.
- Exemplo com MD5:

  `bob: 9f9d51bc70ef21ca5c14f307980a29d8`

  `cob: 386685f06beecb9f35db2e22da429ec9`
- Comparação (mudança de uma letra causa 50% de bits alterados):

```
MD5(bob), bytes  1-8: 1001111110011101010100011011110001110000111011110010000011100101 0
MD5(cob), bytes  1-8: 0011100001100110100001011111000000110101111101110110010111001111 1

MD5(bob), bytes 9-16: 0101110000010100111100110000011110011000000001010001010011101100 0
MD5(cob), bytes 9-16: 0011010111011011001011100010000101101101001000010100111101100100 1
```

Hashing - Hashes de Senhas e Sal

- Um dos usos mais comuns para hashes é no armazenamento de senhas:
  - Quando você se registra em um site, sua senha passa por hashing e este valor, denominado H(hash), é armazenado.
  - Mais tarde, quando você faz o login, o envia da senha é a chamada “proposta”: você está propondo que esta é sua senha verdadeira e o servidor deve checar isso.
  - O servidor então checa se H(proposta) = H(senha).
- Agora imagine que as senhas foram roubadas e foi encontrada a seguinte entrada: `5f4dcc3b5aa765d61d8327deb882cf99`. Você consegue adivinhar a senha?
- Para evitar obtenção de senhas a partir de consultas de tabela pré-calculadas, podemos adicionar o sal. O sal é um valor conhecido publicamente que é misturado com a senha antes do hashing. Este valor deve ser ser único e suficientemente longo.
- A senha anterior pode ser corretamente armazenada como (usando SHA-256)
  `cei6LtJVQYSM+n6Cty0O2w==`,
  `bd51dac1e2fca8456069f38fcce933f1ff30a656320877b5 96a14a0e05db9567`

Hashing - Hashes de Senhas no Python

- O Python possui a biblioteca `scrypt`, cujo algoritmo é descrito no RFC 7914. Outras opções são os módulos `bcrypt` e `Argon2`.
- O seguinte código deriva a `key` (hash) para ser armazenado em arquivo:

```
import os
from cryptography.hazmat.primitives.kdf.scrypt import Scrypt
from cryptography.hazmat.backends import default_backend

salt = os.urandom(16)

kdf = Scrypt(salt=salt, length=32,
        n=2**14, r=8, p=1,
        backend=default_backend())

key = kdf.derive (b"my great password")
```

- Para verificar se uma senha está correta, pode-se escrever:

```
kdf = Scrypt(salt =salt, length =32,
        n=2**14, r=8, p=1,
        backend=default_backend())
kdf.verify(b"my great password", key)
print("Success! (Exception if mismatch)")
```

Referências

Referências - Aula 8

- Capítulos 1, 2 e 3 do livro “Practical Cryptography in Python: Learning Correct Cryptography by Example” de Seth James Nielson e Christopher K. Monson.

The End!