Funções de dispersão criptográfica

Prof. Dieisson Martinelli
dieisson.martinelli@udesc.br

Programa

- Introdução
- Propriedades
- Aplicações
- Exemplos de funções *hash* atuais
- Utilizando *hash* em Python
- Atividades

Introdução

- Uma **função de dispersão criptográfica** (*hash* criptográfica) é uma função *hash* considerada **não-inversível**, ou seja, ela é praticamente impossível de obter o valor de entrada a partir do valor de dispersão
  - Ela aceita uma **mensagem M** de **tamanho variável** e produz uma **saída de tamanho fixo**:

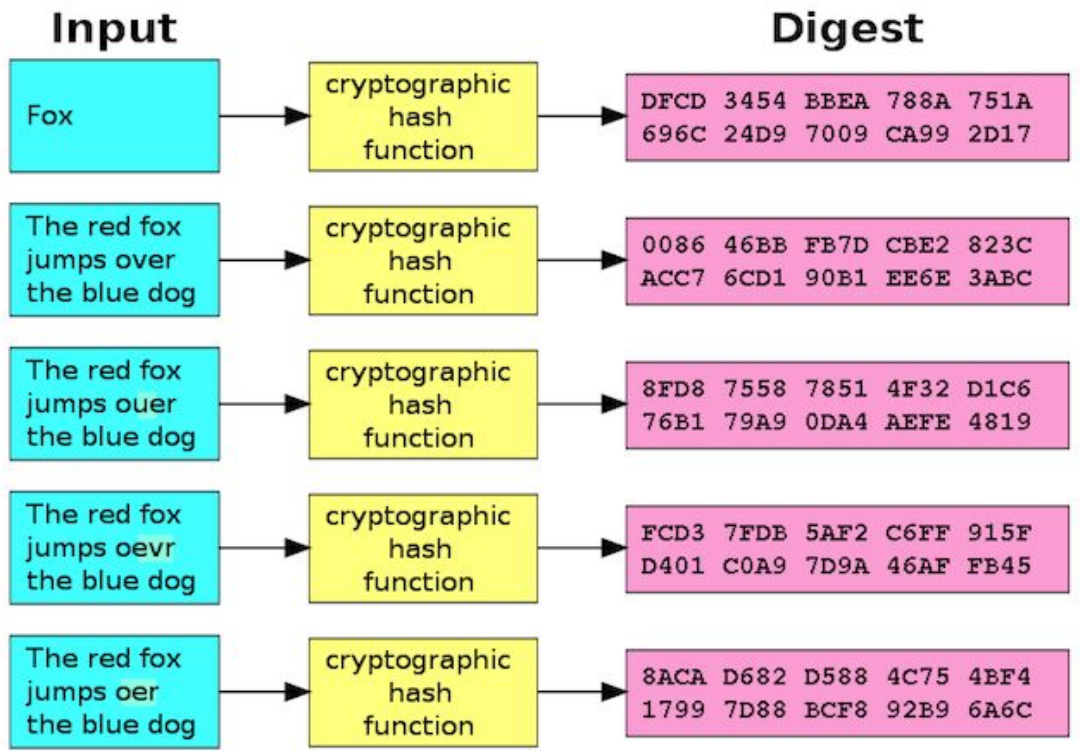
$$\mathbf{h = H(M)}$$

  - A **saída da função H** é chamada **código *Hash*** ou de **resumo** (*digest*) da mensagem (*message digest*)

Introdução

- **Comprimento fixo de saída** independente do tamanho da entrada

over
ouer
oevr
oer



Introdução

Características:

- Uma função de dispersão criptográfica **não necessita de uma chave** para gerar uma saída
- Fornece a **capacidade de detecção de erro**, pois a **mudança** em um um **único bit** da mensagem **altera completamente** seu **resumo**
- O **objetivo** de uma função de dispersão criptográfica é **produzir** algo semelhante a uma **impressão digital** de um arquivo, mensagem ou qualquer outro bloco de dados
  - Normalmente o **código *hash*** é adicionado à **mensagem na origem**. No destino o **código *hash* é recalculado** e caso seja igual ao enviado pela origem, a **mensagem é considerada autêntica**

Propriedades

- Uma função de dispersão criptográfica deve possuir **quatro propriedades principais**:
  - **Fácil de calcular:** Deve ser simples gerar o hash (resumo) de uma mensagem, ou seja, calcular **h = H(M).**
  - **Difícil de inverter (unidirecional)**: Não deve ser possível descobrir a mensagem original apenas olhando para o **hash h**.
  - **Resistente a alterações:** Se mudarmos qualquer parte da mensagem, o hash também deve mudar. Assim, não dá para enganar o sistema alterando a mensagem e mantendo o mesmo resumo.
  - **Sem colisões:** Deve ser muito difícil encontrar duas mensagens diferentes que gerem o mesmo hash. Isso é chamado de resistência a colisões.

Propriedades

- Uma **função de dispersão criptográfica** com todos os critérios listados anteriormente ainda pode possuir propriedades indesejadas
- Atualmente, funções de dispersão criptográficas populares estão **vulneráveis a certos ataques**
  - Em 2004 foram descobertas fraquezas em um número de funções *hash* bastante populares na época (p. ex. SHA-0, RIPEMD e MD5)
- Mesmo que uma função hash nunca tenha sido quebrada, um **ataque bem sucedido** a uma **variante mais fraca**, pode levar a perda de confiança e o abandono da função

Aplicações

- Possuem várias **aplicações**:
  - **Assinatura digital**
  - Algoritmos de **autenticação de mensagens** (*message authentication code - MAC*)
  - **Autenticação de usuários** (*username* e *password*)
  - Como ***checksum*** para detectar **corrupção acidental de dados** (p. ex. em um download de um arquivo)

Aplicações

- **Verificação da integridade** de um arquivo, para comparar se houve ou não alteração
- **Algoritmo de prova-de-trabalho** (*proof-of-work*): para exigir que o solicitante do trabalho/serviço primeiramente resolva uma *hash* que consuma tempo de execução razoável, para evitar ataques de DoS (negação de serviços); ou para validação de criptomoedas (ex. para verificar transações)
- **Geradores de números pseudoaleatórios** e **derivadores de chave**: a partir de uma chave segura (ex. do *internet banking*), gerar outras chaves com o *hash* (p. ex. um número de *token*)

Exemplos de *hashes* usados atualmente

- MD5: produz um valor de *hash* de **128 bits** expresso em **32 caracteres** (hex.). Não é utilizado para criptografia devido à vulnerabilidades, mas é usado para **verificação de integridade** contra corrupção não intencional de dados
- SHA: consistem em uma **família de algoritmos** denominados de *Secure Hash Algorithm* (SHA-0, SHA-1, SHA-2, SHA-3 e variações; com saídas entre 160 e 512 bits), publicados pela *National Institute of Standards and Technology* (NIST)
  - SHA-2 é implementada em algumas aplicações de segurança e protocolos amplamente usados, incluindo TLS e SSL, SSH e IPsec

Utilizando funções *hash* em Python

- **Módulo *hashlib***: implementa uma interface comum para muitos algoritmos diferentes de *hash* seguro e resumo de mensagens
  - Estão incluídos os **algoritmos de *hash* seguros** FIPS SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 (definidos no padrão FIPS 180-4), a série SHA-3 (no padrão FIPS 202), bem como o algoritmo MD5 da RSA (definido na RFC 1321)
- Existe um **método construtor** nomeado para cada tipo de *hash*. Todos retornam um objeto *hash* com a mesma interface simples
  - Por exemplo: use sha256() para criar um objeto hash SHA-256.

Utilizando funções *hash* em Python

- Exemplo 1 – **sha256()**:

```
import hashlib

string = "Estruturas de Dados II"
m = hashlib.sha256()
m.update(string.encode())

print(m.hexdigest())

#saída: 687def7549d4a34553181c733ffa7cd76877b9739019d34e05de57c3a02bc919
```

  - encode() é necessário porque o método opera com um objeto de bytes, que também pode ser criado com o prefixo “b” na *string*:
    - string = b"Estrutura de Dados II"

Utilizando funções *hash* em Python

- Exemplo 2 – **construtor new()**:

```
import hashlib

m = hashlib.new('md5')
m.update(b"Estruturas de Dados II")

print(m.hexdigest())
```

```
import hashlib

m = hashlib.new('sha512')
m.update(b"Estruturas de Dados II")

print(m.hexdigest())
```

Utilizando funções *hash* em Python

- Exemplo 3 – **várias hashes**:

```
import hashlib

data = {'md5', 'sha1', 'sha224', 'sha256', 'sha384', 'sha512',
        'sha3_224', 'sha3_256', 'sha3_384', 'sha3_512'}

for i in data:
    m = hashlib.new(i)
    m.update(b"Estruturas de Dados II")
    print(i, ":", m.hexdigest())
```

  - Mostra todas as *hashes* dos métodos definidos no conjunto *data* para a frase "`Estruturas de Dados II`"

Utilizando funções *hash* em Python

- Exemplo 4 – arquivo ***hash_file.py***:

```
import sys
import hashlib

BUF_SIZE = 65536
md5 = hashlib.md5()
sha1 = hashlib.sha1()

with open(sys.argv[1], 'rb') as f:
    while True:
        data = f.read(BUF_SIZE)
        if not data:
            break
        md5.update(data)
        sha1.update(data)
print("MD5: {0}".format(md5.hexdigest()))
print("SHA1: {0}".format(sha1.hexdigest()))
```

Utilizando funções *hash* em Python

- Outros métodos e atributos da *hashlib*:
  - hashlib.algorithms_available
    - Retorna os nomes dos algoritmos de *hash* que estão disponíveis no interpretador Python em execução
  - hashlib.algorithms_guaranteed
    - Retorna os nomes dos algoritmos de *hash* com suporte garantido em todas as plataformas
  - hash.name
    - Retorna o nome do algoritmo de *hash* utilizado pelo objeto *hashlib*
  - hash.copy
    - Retorna uma cópia (“clone”) do objeto *hash*

Atividades

- Teste os exemplo 1, 2, 3 e os métodos *copy* e *name* do *hashlib*
- Teste o exemplo 4 (*hash_file.py*) para calcular o *checksum* de um arquivo passado como argumento
- Teste o exemplo 5 para o MD5 e compare o *hash* obtido com o programa em Java e o *hash* obtido com o script em Python

Atividades

- **Desafio**: escreva um programa que decifre uma senha de 8 caracteres. Esta senha possui apenas os seguintes caracteres:
  - 1 2 3 a e i o u

- A hash MD5 que representa a senha é:
  - bc9b5d33aca080fda8b85032bc0ed1ea