Verificação de Integridade de Arquivos (MD5, SHA-1, SHA-256, CheckSum)

QXD0099 - Desenvolvimento de Software para Persistência

Universidade Federal do Ceará - *Campus* Quixadá

Prof. Francisco Victor da Silva Pinheiro
victorpinheiro@ufc.br

Agenda

- Introdução à Verificação de Integridade
  - Como funciona?
  - Exemplos de Aplicação
- Principais Algoritmos de Hash
  - MD5 (Message Digest 5)
  - SHA-1 (Secure Hash Algorithm 1)
  - SHA-256 (Secure Hash Algorithm 256 bits)
  - CheckSum
  - Tabela Comparativa
- Ferramentas e Exemplos Práticos
- Códigos Python para Verificação de Integridade

Introdução à Verificação de Integridade

- A verificação de integridade é um processo usado para garantir que os dados ou arquivos não foram alterados desde sua criação.

- **Usado em:**
  - Verificação de downloads.
  - Comparação de backups.
  - Segurança em transferências de dados.

Introdução à Verificação de Integridade

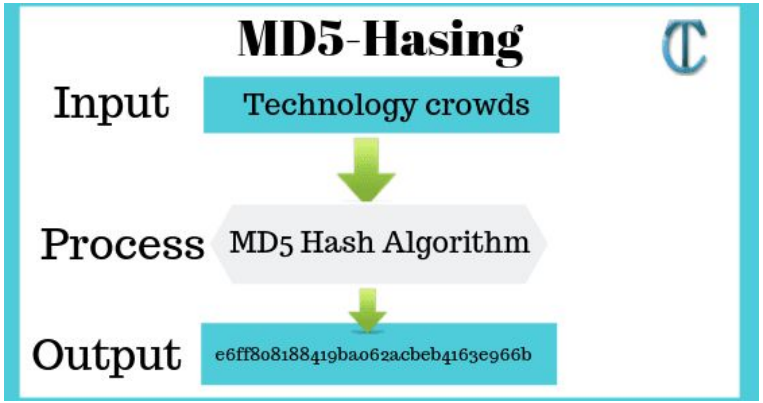
- **Como funciona?**
  - Um algoritmo de hash gera uma assinatura única para o arquivo (o hash).
  - Se o arquivo for alterado, o hash gerado será diferente.

- **Exemplos de Aplicação**
  - Verificar a integridade de software baixado.
  - Garantir que dados transmitidos não foram corrompidos.

Principais Algoritmos de Hash

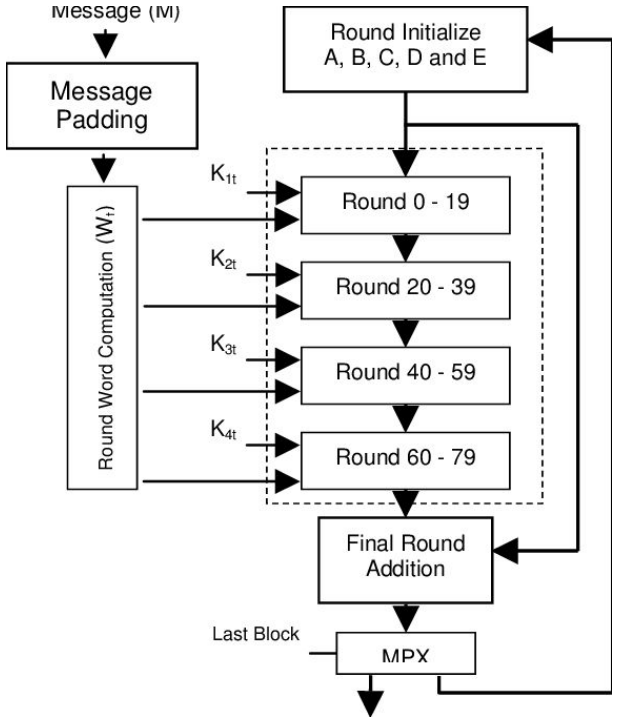
- **MD5 (Message Digest 5)**
  - Muito usado no passado para verificar integridade.

- Vantagem: rápido e amplamente suportado.
- Desvantagem: vulnerável a ataques de colisão.



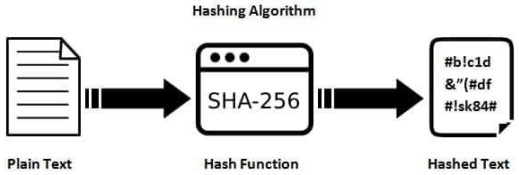
Principais Algoritmos de Hash

- **SHA-1 (Secure Hash Algorithm 1)**
  - Substituto do MD5, mas também considerado inseguro para aplicações críticas.
- Vantagens: O SHA-1 era mais seguro que o MD5.
- Desvantagens: O SHA-1 era vulnerável a ataques de colisão, que permitem que dois arquivos diferentes gerem hashes idênticos. Isso possibilita substituir um arquivo pelo outro e gerar um certificado fraudulento.



Principais Algoritmos de Hash

- **SHA-256 (Secure Hash Algorithm 256 bits)**
  - Parte da família SHA-2.
  - Usado em certificados digitais e blockchain.
- Vantagens
  - É um algoritmo altamente seguro e amplamente utilizado.
  - É uma forma relativamente simples de assegurar integridade e rapidez
- Desvantagens
  - Não é compatível com sistemas ou softwares mais antigos que usam MD5 ou SHA-1
  - Exige cálculos mais complexos do que algoritmos mais antigos, o que o torna mais lento e exige mais poder de processamento



Principais Algoritmos de Hash

- **CheckSum**
- Soma de verificação simples.
- Compara valores antes e depois de transferências de dados, mas é menos seguro do que algoritmos de hash.
- O CheckSum só exige adição e o processamento necessário para criar ou verificar um CheckSum é pequeno.
- A maioria das redes que empregam uma técnica de CheckSum usam um CheckSum de 16 ou 32 bits e geram CheckSum único para o pacote inteiro.
- Tem a desvantagem de não detectar todos os erros comuns.

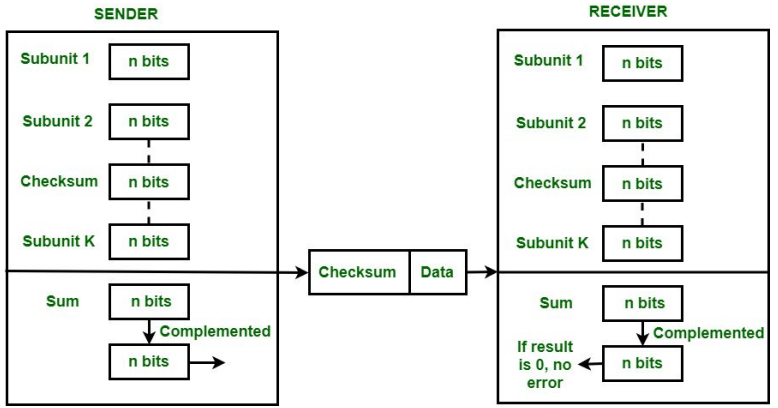


Tabela Comparativa

Algoritmo	Comprimento do Hash	Velocidade	Segurança Atual
MD5	128 bits	Alta	Baixa
SHA-1	128 bits	Média	Baixa
SHA-256	128 bits	Baixa	Alta
CheckSum	Variável	Alta	Muito Baixa

Ferramentas e Exemplos Práticos

- **Ferramentas Comuns**
  - Windows: certutil
  - Linux/Mac: md5sum, sha256sum
  - Online: Sites que calculam hash a partir de uploads de arquivos.
    - https://hash.online-convert.com/pt

- **Exemplo 1: Verificar Hash de um Arquivo no Linux**
  - # Gerar hash SHA-256 de um arquivo
    - sha256sum arquivo.txt

- **Exemplo 2: Comparar Hash no Windows**
  - certutil -hashfile arquivo.zip SHA256

Códigos Python para Verificação de Integridade

- **MD5 (128 bits)**
- Utiliza a biblioteca hashlib, que oferece suporte a algoritmos de hash padrão.
- A função hexdigest() retorna o hash em formato hexadecimal.

```python
import hashlib

# Caminho para o arquivo a ser verificado
file_path = 'arquivo.txt'

# Gerar hash MD5
with open(file_path, 'rb') as f:
    file_data = f.read()
    md5_hash = hashlib.md5(file_data).hexdigest()

print(f"Hash MD5: {md5_hash}")
```

Códigos Python para Verificação de Integridade

- **MD5 (128 bits)**
- Verificação de integridade

```python
import hashlib

# Caminho para o arquivo a ser verificado
file_path = 'md5.txt'

# Hash MD5 esperado (gerado anteriormente e armazenado)
expected_md5_hash = '8e506a437730815e2574c35dcadfccbe'

# Gerar o hash MD5 do arquivo atual
def calculate_md5(file_path):
    with open(file_path, 'rb') as f:
        file_data = f.read()
        return hashlib.md5(file_data).hexdigest()

# Comparar o hash calculado com o esperado
calculated_md5_hash = calculate_md5(file_path)
print(f"Hash MD5 Calculado: {calculated_md5_hash}")
print(f"Hash MD5 Esperado: {expected_md5_hash}")

if calculated_md5_hash == expected_md5_hash:
    print("Integridade verificada: o arquivo é autêntico.")
else:
    print("Integridade comprometida: o arquivo foi alterado ou corrompido.")
```

Códigos Python para Verificação de Integridade

- **SHA-1 (160 bits)**

```python
import hashlib

# Caminho para o arquivo a ser verificado
file_path = 'arquivo.txt'

# Gerar hash SHA-1
with open(file_path, 'rb') as f:
    file_data = f.read()
    sha1_hash = hashlib.sha1(file_data).hexdigest()

print(f"Hash SHA-1: {sha1_hash}")
```

Códigos Python para Verificação de Integridade

- **SHA-256 (256 bits)**

```python
import hashlib

# Caminho para o arquivo a ser verificado
file_path = 'arquivo.txt'

# Gerar hash SHA-256
with open(file_path, 'rb') as f:
    file_data = f.read()
    sha256_hash = hashlib.sha256(file_data).hexdigest()

print(f"Hash SHA-256: {sha256_hash}")
```

Códigos Python para Verificação de Integridade

- **CheckSum**
  - (Soma de Verificação)

```python
import hashlib

# Caminho dos arquivos
file_path = 'arquivo.txt'
checksum_file_path = 'checksum.txt'

# Função para calcular o checksum usando hashlib (SHA-256 por padrão)
def calculate_checksum(file_path, hash_algorithm='sha256'):
    try:
        hash_function = hashlib.new(hash_algorithm)
        with open(file_path, 'rb') as file:
            while chunk := file.read(8192):
                hash_function.update(chunk)
        return hash_function.hexdigest()
    except FileNotFoundError:
        print(f"Erro: O arquivo '{file_path}' não foi encontrado.")
        return None

# Calcular o checksum e salvar no arquivo
checksum_value = calculate_checksum(file_path)
if checksum_value:
    with open(checksum_file_path, 'w') as checksum_file:
        checksum_file.write(checksum_value)
    print(f"Checksum salvo em '{checksum_file_path}'.")
```

Códigos Python para Verificação de Integridade

- **CheckSum**
  - Verificação de integridade

```python
import hashlib

# Caminhos dos arquivos
file_path = 'arquivo.txt'  # Arquivo a ser verificado
checksum_file_path = 'checksum.txt'  # Arquivo contendo o checksum esperado

# Função para calcular o checksum de um arquivo
def calculate_checksum(file_path, hash_algorithm='sha256'):
    try:
        # Seleciona o algoritmo de hash
        hash_function = hashlib.new(hash_algorithm)

        # Leitura do arquivo em blocos para eficiência
        with open(file_path, 'rb') as file:
            while chunk := file.read(8192):
                hash_function.update(chunk)

        return hash_function.hexdigest()
    except FileNotFoundError:
        print(f"Erro: O arquivo '{file_path}' não foi encontrado.")
        return None
```

```python
# Função para ler o checksum esperado
def read_expected_checksum(checksum_file_path):
    try:
        with open(checksum_file_path, 'r') as checksum_file:
            return checksum_file.read().strip()
    except FileNotFoundError:
        print(f"Erro: O arquivo de checksum '{checksum_file_path}' não foi encontrado.")
        return None

# Função para salvar o checksum calculado em um arquivo
def save_checksum(checksum, checksum_file_path):
    try:
        with open(checksum_file_path, 'w') as checksum_file:
            checksum_file.write(checksum)
        print(f"Checksum salvo em '{checksum_file_path}'.")
    except Exception as e:
        print(f"Erro ao salvar o checksum: {e}")

# Calcular e comparar checksums
expected_checksum = read_expected_checksum(checksum_file_path)
calculated_checksum = calculate_checksum(file_path)

if calculated_checksum:
    print(f"Checksum Calculado: {calculated_checksum}")
    if expected_checksum:
        print(f"Checksum Esperado: {expected_checksum}")
        if calculated_checksum == expected_checksum:
            print("Integridade verificada: o arquivo é autêntico.")
        else:
            print("Integridade comprometida: o arquivo foi alterado ou corrompido.")
    else:
        print("Checksum esperado não encontrado. Salvando o checksum calculado como referência.")
        save_checksum(calculated_checksum, checksum_file_path)
```

Referências

- https://www.ssldragon.com/pt/blog/algoritmos-hash-sha-1-vs-sha-2/
- https://www.freecodecamp.org/portuguese/news/md5-x-sha-1-x-sha-2-qual-e-o-hash-de-criptografia-mais-seguro-e-como-verifica-lo/
- Bruce Schneie. Cryptanalysis of MD5 and SHA: Time for a New Standard. Computerworld. Flickr's API Signature Forgery Vulnerability Thai Duong and Juliano Rizzo
- Chad Perrin (5 de Dezembro de 2007). Use MD5 hashes to verify software downloads. TechRepublic.

Obrigado!

Dúvidas?

Universidade Federal do Ceará - *Campus* Quixadá

Prof. Francisco Victor da Silva Pinheiro
victorpinheiro@ufc.br