Relatório de Implementação de Algoritmo MD5

Felipe Cartaxo de Freitas Sheila Lima Lee

8 de Julho de 2024

Abstract

Este relatório detalha a implementação de um algoritmo de hashing MD5 em Python. O código é explicado passo a passo, e os resultados são comparados com a biblioteca hashlib do Python para verificar a precisão da implementação.

1 Introdução

Funções de hashing são fundamentais para garantir a integridade e segurança de dados em diversas aplicações. O MD5 (Message Digest Algorithm 5) é uma dessas funções amplamente utilizadas, apesar de suas vulnerabilidades conhecidas. Este relatório descreve a implementação de um algoritmo MD5 em Python, abordando cada parte do código e discutindo os resultados obtidos.

2 Metodologia

2.1 Constantes e Rotações

O algoritmo utiliza tabelas predefinidas de constantes e valores de rotação para cada um dos 64 passos de processamento. As constantes são derivadas da função seno, conforme especificado no padrão MD5.

```
1 import math
2
3 rotate_by = [7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12,
      17, 22,
```

```
            5,  9, 14, 20, 5,  9, 14, 20, 5,  9, 14, 20, 5,  9,
    14, 20,
            4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11,
    16, 23,
            6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10,
    15, 21]

constants = [int(abs(math.sin(i + 1)) * 4294967296) & 0xFFFFFFFF
    for i in range(64)]
```

2.2 Função de Padding

A função de padding ajusta o comprimento da mensagem para que seja múltiplo de 512 bits, conforme necessário pelo algoritmo MD5.

```
def pad(msg):
    msg_len_in_bits = (8 * len(msg)) & 0xffffffffffffffff
    msg.append(0x80) # Adiciona 1 bit seguido por 0 bits at  o
    comprimento ser congruente a 448 (mod 512)

    while len(msg) % 64 != 56:
        msg.append(0)

    msg += msg_len_in_bits.to_bytes(8, byteorder='little')

    return msg
```

2.3 Buffer Inicial

O buffer inicial do MD5 é especificado na RFC e consiste de quatro valores de 32 bits.

```
init_MDBuffer = [0x67452301, 0xefcdab89, 0x98badcfe, 0x10325476]
```

2.4 Função de Rotação e Processamento

A função de rotação à esquerda é utilizada durante o processamento de cada bloco de 512 bits da mensagem. A função de processamento aplica as operações do MD5 em cada bloco da mensagem.

```
def leftRotate(x, amount):
    x &= 0xFFFFFFFF
    return (x << amount | x >> (32 - amount)) & 0xFFFFFFFF
```

```python
def processMessage(msg):
    init_temp = init_MDBuffer[:]

    for offset in range(0, len(msg), 64):
        A, B, C, D = init_temp
        block = msg[offset: offset + 64]

        for i in range(64):
            if i < 16:
                func = lambda b, c, d: (b & c) | (~b & d)
                index_func = lambda i: i
            elif i < 32:
                func = lambda b, c, d: (d & b) | (~d & c)
                index_func = lambda i: (5 * i + 1) % 16
            elif i < 48:
                func = lambda b, c, d: b ^ c ^ d
                index_func = lambda i: (3 * i + 5) % 16
            else:
                func = lambda b, c, d: c ^ (b | ~d)
                index_func = lambda i: (7 * i) % 16

            F = func(B, C, D)
            G = index_func(i)
            to_rotate = A + F + constants[i] + int.from_bytes(block[4 * G: 4 * G + 4], byteorder='little')
            newB = (B + leftRotate(to_rotate, rotate_by[i])) & 0xFFFFFFFF

            A, B, C, D = D, newB, B, C

        for i, val in enumerate([A, B, C, D]):
            init_temp[i] += val
            init_temp[i] &= 0xFFFFFFFF

    return sum(buffer_content << (32 * i) for i, buffer_content in enumerate(init_temp))
```

2.5 Conversão para Hexadecimal

A função final converte o digest calculado para um formato hexadecimal legível.

```python
def MD_to_hex(digest):
    raw = digest.to_bytes(16, byteorder='little')
    return '{:032x}'.format(int.from_bytes(raw, byteorder='big'))
```

2.6 Função Principal

A função principal reúne todos os passos para calcular o hash MD5 de uma mensagem.

```
def hashMD5Aluno(msg):
    msg = bytearray(msg, 'ascii')
    msg = pad(msg)
    processed_msg = processMessage(msg)
    message_hash = MD_to_hex(processed_msg)

    return message_hash
```

2.7 Testes

Os resultados do algoritmo implementado são comparados com os da biblioteca hashlib para verificar a precisão.

```
import hashlib

print("=== Testando MD5 com a hashlib ===")
print(hashlib.md5("teste".encode('utf-8')).hexdigest(), "\n")

print("=== Testando MD5 com meu algoritmo ===")
hashdoAluno = hashMD5Aluno('teste')
print(hashdoAluno)
```

3 Resultados e Discussão

Os testes mostram que a implementação do algoritmo MD5 produz resultados idênticos aos da biblioteca hashlib do Python, confirmando a precisão do algoritmo implementado.

```
=== Testando MD5 com a hashlib ===
698dc19d489c4e4db73e28a713eab07b

=== Testando MD5 com meu algoritmo ===
698dc19d489c4e4db73e28a713eab07b
```

4 Conclusão

Este relatório detalhou a implementação de um algoritmo MD5 em Python. Através dos testes realizados, foi possível verificar que a implementação é precisa e comparável aos resultados da biblioteca hashlib. Este estudo reforça

a importância de entender os fundamentos dos algoritmos de hashing para aplicações em segurança da informação.

5 Referências

- MD5 Algorithm — What Is MD5 Algorithm? — MD5 Algorithm Explained — Network Security — Simplilearn. Disponível em: `https://www.youtube.com/watch?v=r6GlzIWiMD0`
- RFC 1321 - The MD5 Message-Digest Algorithm. Disponível em: `https://datatracker.ietf.org/doc/html/rfc1321?authuser=1`