Sprawozdanie Podstawy Kryptografii- Funkcje Skrótu

Michał Puńko 155863

1. Screenshoty implementacji

```python
def md5(tekst):
    return hashlib.md5(tekst).hexdigest()

def sha1(tekst):
    return hashlib.sha1(tekst).hexdigest()

def sha256(tekst):
    return hashlib.sha256(tekst).hexdigest()

def sha512(tekst):
    return hashlib.sha512(tekst).hexdigest()

def sha3(tekst):
    return hashlib.sha3_512(tekst).hexdigest()

def generujplik(nazwapliku, rozmiarmb):
    with open(nazwapliku, 'wb') as f:
        f.write(os.urandom(rozmiarmb * 1024 * 1024))

def czytajplik(nazwapliku):
    with open(nazwapliku, 'rb') as f:
        return f.read()
```

```python
def policzkolizje(hashe):
    widziane = set()
    kolizje = 0
    for h in hashe:
        prefiks = h[:3]
        if prefiks in widziane:
            kolizje += 1
        else:
            widziane.add(prefiks)
    return kolizje

def sac_test(funkcjaskrotu, dane):
    oryginalny_hash = funkcjaskrotu(dane)
    oryginalne_bity = bin(int(oryginalny_hash, 16))[2:].zfill(len(oryginalny_hash)*4)  # Hex na bity
    zmienione_bity = 0
    for i in range(len(dane)):
        dane_zmienione = bytearray(dane)
        dane_zmienione[i] ^= 1
        nowy_hash = funkcjaskrotu(bytes(dane_zmienione))
        nowe_bity = bin(int(nowy_hash, 16))[2:].zfill(len(nowy_hash)*4)
        zmienione_bity += sum(a != b for a, b in zip(oryginalne_bity, nowe_bity))
    return zmienione_bity / (len(dane) * len(oryginalne_bity))
```

2. Omówienie sposobu implementacji

Aplikacja została napisana w języku Python z wykorzystaniem biblioteki hashlib. Obsługuje następujące algorytmy skrótu:
- MD5
- SHA-1
- SHA-256
- SHA-512
- SHA3-512

Funkcjonalności:
- Obliczanie skrótów dla danych wejściowych (tekst/plik).
- Pomiar czasu wykonania hashy dla plików o rozmiarach 1MB, 5MB i 10MB.
- Analiza liczby kolizji (na pierwszych 12 bitach skrótu).
- Test SAC – analiza wpływu zmiany pojedynczego bitu na wynikowy hash.
- Generowanie wykresów porównawczych.

Kod umożliwia dokładne porównanie bezpieczeństwa i wydajności każdej funkcji skrótu.

3. Określenie roli soli w tworzeniu skrótów

Sól (salt) to losowy ciąg znaków dodawany do danych wejściowych przed wykonaniem operacji skrótu. Jej zastosowanie:
- Chroni przed atakami słownikowymi
- Powoduje, że identyczne hasła nie będą miały identycznych hashy.
- Zwiększa entropię haseł, szczególnie krótkich.

Przykład:

Bez soli:
hash("password") → 5f4dcc3b5aa765d61d8327deb882cf99

Z solą "abc":
hash("abcpassword") → e99a18c428cb38d5f260853678922e03

4. Odpowiedź oraz jej uzasadnienie na pytanie postawione w pkt. 4

Nie, MD5 nie spełnia obecnych standardów bezpieczeństwa.

Powody:
- Zostały odkryte praktyczne kolizje – możliwe jest znalezienie różnych danych dających ten sam hash.
- Algorytm jest szybki, co paradoksalnie czyni go bardziej podatnym na ataki brute-force.

Wniosek: MD5 nie powinien być stosowany w kontekście bezpieczeństwa, np. do przechowywania haseł lub walidacji ważnych danych.

5. Zestawienie uzyskanych wyników wraz ze stosownymi wnioskami

Liczba kolizji (SHA-512, 12 pierwszych bitów): 120 kolizji na 1000 prób

Współczynnik SAC (SHA-512): 0.4984 – bliski ideałowi 0.5

Porównanie czasów haszowania:

Rozmiar 1MB:

Algorytm	Czas [s]
MD5	0.001017
SHA-1	0.000400
SHA-256	0.000413
SHA-512	0.000864
SHA3-512	0.002760

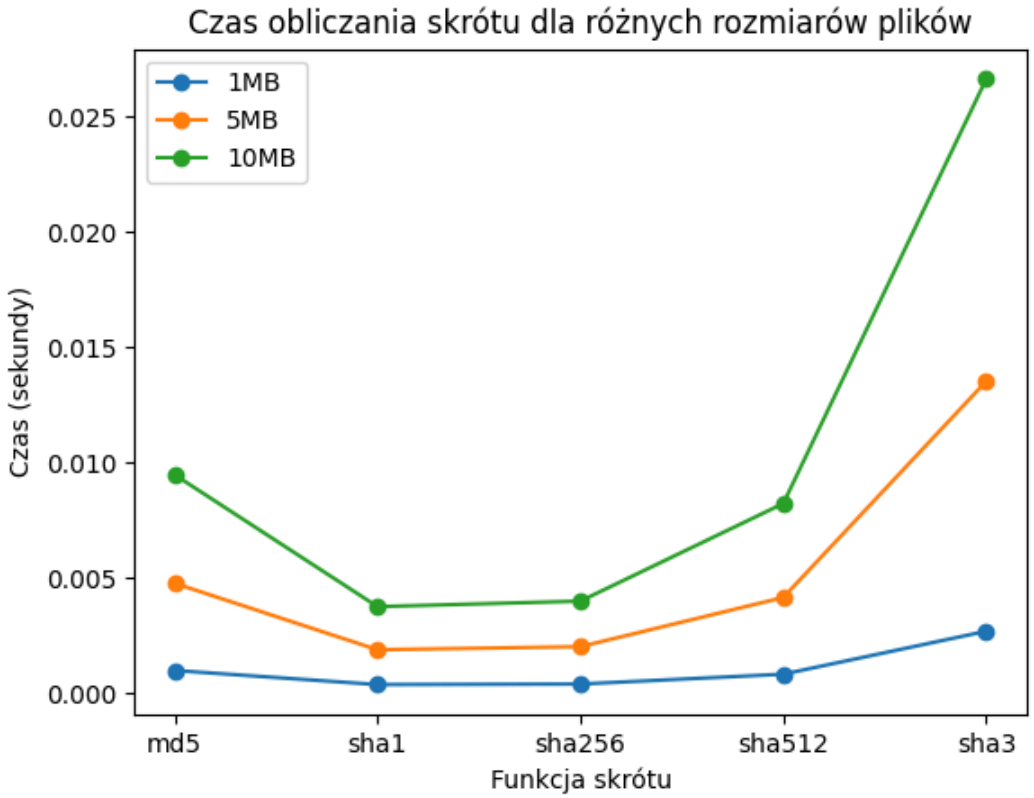
Rozmiar 5MB:

Algorytm	Czas [s]
MD5	0.004861
SHA-1	0.001928
SHA-256	0.002052
SHA-512	0.004247
SHA3-512	0.013717

Rozmiar 10MB:

Algorytm	Czas [s]
MD5	0.009642
SHA-1	0.003859
SHA-256	0.004119
SHA-512	0.008492
SHA3-512	0.027279

Wizualizacja:



Wnioski:

- SHA-1 i SHA-256 są najszybsze przy małych i średnich plikach.
- SHA-512 oferuje wyższe bezpieczeństwo przy akceptowalnym czasie działania.

- SHA3-512 jest najwolniejszy, ale oferuje niezależną od SHA-2 konstrukcję i bardzo silne właściwości kryptograficzne.
- MD5, mimo dobrej wydajności, nie spełnia wymagań bezpieczeństwa i nie powinien być stosowany w nowych systemach.
- Współczynnik SAC i liczba kolizji potwierdzają poprawne właściwości losowości i dyfuzji SHA-512.