Artykuł o algorytmach kryprograficznych

Misiak Adrian, Moskal Michał

*Omówienie problemu
Algorytmy haszujace sa wykorzysytwane do mapowania danych różnych rozmiarów na stałe długościowe wartośći, zwane haszami. Rozwiazuje one wiele problemó, w tym:

- Unikalność: Pomagaja w identyfikowaniu unikalnych elementó w dużych zbiorach danych.
- Szybkie wyszukiwanie: Umożliwiaja szybkie wyszukiwanie danych, ponieważ operacje na haszach sa efektywne.
- Zabezpieczenie: Stosowane sa do zabezpieczenia danych poprzez generowanie haszy haseł, zapobiegajac w ten sposób odczytaniu oryginalnej wartości.

Przykładem zastosowania algorytmów haszujacych jest mechanizm przechowywania haseł w bazah danych, gdzie zamiast przechowywać hasło w formie tekstowej, przechowuje sie jego hasz. Dzieki temu nawet jeśli baza danych zostanie naruszona, atakujacy nie bedzie w stanie odczytać oryginalnych haseł.

Omówienie kilku algorytmów

B-Crypt

- **Zastosowanie:** B-Crypt jest stosowany głównie do bezpiecznego haszowania haseł w systemach uwierzytelniania.
- **Działanie:** B-Crypt jest oparty na funkcji haszujacej opartej na kluczach. Używa tajnego klucza do haszowania danych, co zapewnia silne zabezpieczenia.
- **Złożonośc haszowania:** $O(2^k), gdziektozłożonośćpracy, określajacailerundhaszowaniazostaniew$

Przykładowa implementacja (w jezyku Python, wykorzystujac biblioteke bcrypt):

```
import bcrypt

//Hashowanie hasla
password = b"my_password"
hashed_password = bcrypt.hashpw(password, bcrypt.gensalt())
```

```
print(hashed_password)

//Weryfikacja hasla
entered_password = b"my_password"
if bcrypt.checkpw(entered_password, hashed_password):
    print("Password match")
else:
    print("Password does not match")
```

RSA (Rivest-Shamir-Adleman)

- **Zastosowanie:** RSA jest stosowany do szyfrowania i podpisywania cyfrowego w celu zapewnienia bezpiecznej lomunikacji internetowej, takiej jak HTTPS.
- **Działanie:** RSA opiera sie na problemie fakrotyzacji dyżych liczb pierwszych. Generuje sie nim dwa klucze: publiczny i prywatny. Klucz publiczny jest używany do szyfrowania danych, podczas gdy klycz prywatny używany jest do ich deszyfrowania.

- **Złożonośc haszowania:**Generowanie kluczy: O(n^3), $gdzientodlugośćklucza.GenerowaniekluczywRSAwy$
Przykładowa implementacja (w jezyku Python, wykorzystujac biblioteke Crypto):

```
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP

//Generowanie kluczy
key = RSA.generate(2048)
private_key = key.export_key()
public_key = key.publickey().export_key()

//Szyfrowanie danych
cipher = PKCS1_OAEP.new(RSA.import_key(public_key))
encrypted_data = cipher.encrypt(b"Hello, world!")
print(encrypted_data)

//Deszyfrowanie danych
cipher = PKCS1_OAEP.new(RSA.import_key(private_key))
decrypted_data = cipher.decrypt(encrypted_data)
print(decrypted_data)
```

AES (Advanced Encryption Standard)

- **Zastosowanie:** AES jest jednym z najczesciej używanych algorytmów szyfrowania symetrycznego. Jest szeroko stosowany w zabezpieczeniu danych, takich jak hasła, karty kredytowe czy poufne dokumenty.
- **Działanie:** AES działa na blokach danych o stałej długości i wykorzystuje klucz do szyfrowania i deszyfrowania danych.

- **Złożonośc haszowania:**Złożoność szyfrowania: O(1) - złożoność stała, ponieważ szyfrowanie bloku danych wymaga stałej liczby operacji niezależnie od długości danych wejściowych.

Przykładowa implementacja (w jezyku Python, wykorzystujac biblioteke PyCryptodome):

```
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes

//Generowanie klucza
key = get_random_bytes(16)

//Szyfrowanie danych
cipher = AES.new(key, AES.MODE_ECB)
plaintext = b'Hello, world!'
ciphertext = cipher.encrypt(plaintext)
print(ciphertext)

//Deszyfrowanie danych
decrypted_data = cipher.decrypt(ciphertext)
print(decrypted_data)
```

SHA-256 (Secure Hash Algorithm 256-bit)

- **Zastosowanie:** SHA-256 jest czesto stosowanym algorytmem haszowania. Jest używany do generowania wartości skrótu, które można wykorzystać do weryfikacji integralności danych lub do bezpiecznego przechowywania haseł.
- **Działanie:** SHA-256 przekształca dane wejściowe o róznej długośći na stały skrót o długości 256 bitów.
- **Złożonośc haszowania:**Złożoność szyfrowania: O(1) - złożoność stała, ponieważ szyfrowanie bloku danych wymaga stałej liczby operacji niezależnie od długości danych wejściowych.

Przykładowa implementacja (w jezyku Python, wykorzystujac biblioteke hashlib):

```
import hashlib

Generowanie skrótu
data = b'Hello, world!'
hashed_data = hashlib.sha256(data).hexdigest()
print(hashed_data)
```

Diffie-Hellman (DH)

- **Zastosowanie:** Protokół Diffie-Hellman jest używany do bezpiecznego wymiany kluczy kryptograficznych pomiedzy dwoma stronami w sposób, który zapewnia tajność komunikacji.
- **Działanie:** DH umożliwia dwóm stronom, nazywanym Alice i Bobem, uzyskanie wspólnego tajnego klucza, który może być nastepnie wykorzystany do szyfrowania i deszyfrowania komunikatów. Protokół ten opiera sie na matematycznych operacjach modulo.
- **Złożonośc haszowania:** O(1) - generowanie kluczy jest zazwyczaj operacja stałego czasu, ponieważ polega na wyborze losowych liczb w określonym zakresie.

Przykładowa implementacja (w jezyku Python):

```
from Crypto.Util import number

//Wybór parametrów DH
p = number.getPrime(256)
g = 2

//Generowanie kluczy dla Alice i Boba
alice_private_key = number.getRandomRange(1, p - 1)
bob_private_key = number.getRandomRange(1, p - 1)
alice_public_key = pow(g, alice_private_key, p)
bob_public_key = pow(g, bob_private_key, p)

//Obliczanie wspólnego tajnego klucza
alice_shared_secret = pow(bob_public_key, alice_private_key, p)
bob_shared_secret = pow(alice_public_key, bob_private_key, p)

//Sprawdzenie, czy obie strony maja ten sam wspólny tajny klucz
assert alice_shared_secret == bob_shared_secret

print("Shared secret:", alice_shared_secret)
```

Wnioski

- Złożoność obliczeniowa: Każdy algorytm ma inna złożoność obliczeniowa, co ma wpływ na jego wydajność i szybkość działania.
- Różnorodność zastosowań: ALgorytmy sa używane w różnych obszarach kryptografii, od uwierzytelniania haseł po bezpieczna komunikacje internetowa.

- Bezpieczeństwo vs. wydajność: Istnieje kompromis miedzy bezpieczeństwem a wydajnościa.
- Rola kluczy kryptograficznych: Klucze sa kluczowymi elementami w zapewnieniu bezpieczeństwa.
- Dobór algorytmów: Ważne jests dopasowanie algorytmu do konkretnego zastosowania.

Referencje

- https://github.com/pyca/bcrypt/
- https://www.geeksforgeeks.org/hashing-passwords-in-python-with-bcrypt/
- https://pycryptodome.readthedocs.io/en/latest/
- https://docs.python.org/3/library/hashlib.html
- https://pycryptodome.readthedocs.io/en/latest/src/public$_key/dh.html$