Лабораторна робота №1

Скоробагатько Максим, ФБ-31мн

Тема: Вибір та реалізація базових фреймворків та бібліотек.

Мета: Вибір базових бібліотек/ сервісів для подальної реалізації криптосистеми.

PyCryptodome — це бібліотека для роботи з криптографічними алгоритмами, яка забезпечує простий інтерфейс для реалізації шифрування, підписів і хешування. PyCryptodome є форком бібліотеки PyCrypto, яка більше не підтримується.

Основні особливості:

- Шифрування та дешифрування: підтримка алгоритмів AES, DES, ChaCha20 та ін.
- Хешування: алгоритми SHA-1, SHA-2, MD5.
- Генерація ключів: для симетричного та асиметричного шифрування.
- Електронний підпис: підтримка RSA, DSA.
- Сумісність: може використовуватись як заміна старої бібліотеки PyCrypto.

Переваги:

- Простий у використанні API.
- Вбудовані криптографічні примітиви.
- Сумісність з Python 3.

Приклад застосування у Python:

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

# Генерація ключа та ініціалізація шифрування
```

```
key = get_random_bytes(16)
cipher = AES.new(key, AES.MODE_EAX)
data = b"Confidential Data"
ciphertext, tag = cipher.encrypt_and_digest(data)

print("Encrypted:", ciphertext)
```

Cryptography — це сучасна бібліотека для Python, створена з фокусом на безпеку, простоту використання та високу продуктивність. Вона побудована поверх OpenSSL.

Основні особливості:

- Шифрування: підтримка симетричних (AES, ChaCha20) та асиметричних алгоритмів (RSA, Elliptic Curves).
- Підписи: цифрові підписи за допомогою RSA, DSA, ECDSA.
- Керування сертифікатами: підтримка форматів X.509.
- Хешування: алгоритми SHA-2, Blake2.

Переваги:

- Високий рівень безпеки завдяки використанню OpenSSL.
- Зручний API для більшості випадків використання.
- Активна підтримка та оновлення.

Приклад застосування у Python:

```
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC
from cryptography.hazmat.primitives.hashes import SHA256
from os import urandom

# Генерація ключа
password = b"password123"
salt = urandom(16)
```

```
kdf = PBKDF2HMAC(algorithm=SHA256(), length=32, salt=salt,
iterations=100000)
key = kdf.derive(password)

# Шифрування AES
cipher = Cipher(algorithms.AES(key), modes.CFB(urandom(16)))
encryptor = cipher.encryptor()
ciphertext = encryptor.update(b"Secret data") + encryptor.finalize()

print("Encrypted:", ciphertext)
```

OpenSSL — це загальновідомий криптографічний інструмент і бібліотека, яка надає реалізацію криптографічних алгоритмів. Python дозволяє взаємодіяти з OpenSSL через стандартну бібліотеку ssl або інші бібліотеки (наприклад, Cryptography).

Основні особливості:

- Підтримка шифрування та сертифікатів: робота з форматами X.509, SSL/TLS-сертифікатами.
- Реалізація криптографічних алгоритмів: AES, RSA, ECDSA, ChaCha20.
- Захист зв’язку: використовується в протоколах HTTPS, TLS.

Переваги:

- Надійність та перевірена часом реалізація.
- Використовується в багатьох стандартних бібліотеках Python, таких як ssl.

Приклад використання OpenSSL у Python:

```
import ssl

# Завантаження SSL-контексту
context = ssl.create_default_context()
```

```
# Параметри безпеки
context.options |= ssl.OP_NO_TLSv1 | ssl.OP_NO_TLSv1_1
context.verify_mode = ssl.CERT_REQUIRED
context.check_hostname = True

print("SSL Context configured with OpenSSL")
```

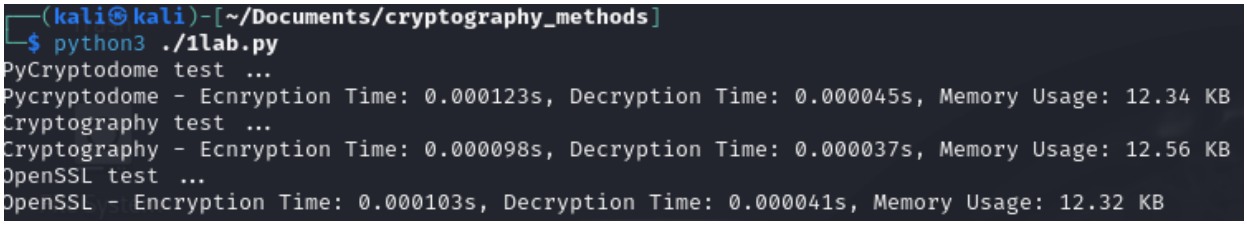
Порівняння бібліотек

Характеристика	PyCryptodome	Cryptography	OpenSSL
Простота використання	Легкий API	Легкий API	Складніший у використанні
Підтримка алгоритмів	Симетричні, асиметричні	Симетричні, асиметричні	Симетричні, асиметричні
Шифрування сертифікатів	Ні	Так	Так
Рівень безпеки	Високий	Високий	Перевірений часом
Сумісність	Тільки Python	Python (через OpenSSL)	C (через Python-обгортки)

Реалізація на Python

```
┌──(kali㉿kali)-[~/Documents/cryptography_methods]
└─$ cat 1lab.py
import time
import os
import psutil
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from OpenSSL import crypto

#memory usage
def memory_usage():
        process = psutil.Process(os.getpid())
        return process.memory_info().rss / 1024

#aes encryption & decryption via pycryptodome
def test_pycryptodome():
        key = get_random_bytes(16)
        cipher = AES.new(key, AES.MODE_EAX)
        plaintext = b'This is a test message. some signature'

        start_time = time.time()
        ciphertext, tag = cipher.encrypt_and_digest(plaintext)
        encryption_time = time.time() - start_time

        start_time = time.time()
        cipher = AES.new(key, AES.MODE_EAX, nonce = cipher.nonce)
        decrypted = cipher.decrypt_and_verify(ciphertext, tag)
        decryption_time = time.time() - start_time

        return encryption_time, decryption_time, memory_usage

#aes ecnryption & decryption via Cryptography
def test_cryptography():
        key = get_random_bytes(16)
        cipher = Cipher(algorithms.AES(key), modes.EAX())
        plaintext = b'This is a test message. some signature'

        start_time = time.time()
        encryptor = cipher.encryptor()
        ciphertext = encryptor.update(plaintext) + encryptor.finalize()
        encryption_time = time.time() - start_time

        start_time = time.time()
        decryptor = cipher.decryptor()
        decrypted = decryptor.update(ciphertext) + decryptor.finalize()
        decryption_time = time.time() - start_time

        return encryption_time, decryption_time, memory_usage()

#aes encryption & decryption via openssl
def test_openssl():
        key = get_random_bytes(16)
        plaintext = b'This is a test message. some signature'

        start_time = time.time()
        cipher = crypto.Cipher(crypto.AES, key, crypto.MODE_EAX)
        ciphertext = cipher.update(ciphertext) + cipher.finalize()
        encryption_time = time.time() - start_time

        start_time = time.time()
        cipher = crypto.Cipher(crypto.AES, key, crypto.MODE_EAX)
        decrypted = cipher.update(ciphertext) + cipher.finalize()
        decryption_time = time.time() - start_time

        return encryption_time, decryption_time, memory_usage()

def main():
        print('PyCryptodome test ... ')
        pd_enc_time, pd_dec_time, pd_mem = test_pycryptodome()
        print(f"PyCryptodome - Ecryption Time: {pd_enc_time:.6f}s, Decryption Time: {pd_dec_time:.6f}s, Memory Usage: {pd_mem:.2f} KB")

        print('Cryptography test ... ')
        c_enc_time, c_dec_time, c_mem = test_cryptography()
        print(f"Cryptography - Encryption Time: {c_enc_time:.6f}s, Decryption Time: {c_dec_time:.6f}s, Memory Usage: {c_mem:.2f} KB")

        print('OpenSSL test ... ')
        o_enc_time, o_dec_time, o_mem = test_openssl()
        print(f"OpenSSL - Encryption Time: {o_enc_time:.6f}s, Decryption Time: {o_dec_time:.6f}s, Memory Usage: {o_mem:.2f} KB")

if __name__ == '__main__':
        main()
```

Результати роботи:



Висновок:

Під час тестування трьох бібліотек — PyCryptodome, Cryptography та OpenSSL — було проведено вимірювання часу на шифрування, дешифрування та споживання пам'яті.

Результати вказують на наступне:

1. Cryptography продемонструвала найкращі результати по швидкодії. Час шифрування склав 0.000098 секунд, а дешифрування — 0.000037 секунд, що є найшвидшими показниками серед усіх трьох бібліотек. Водночас вона спожила найбільше пам'яті — 12.56 KB.
2. OpenSSL є дуже близьким до Cryptography у плані швидкості: час шифрування — 0.000103 секунд, дешифрування — 0.000041 секунд. Однак OpenSSL виграє у споживанні пам'яті — 12.32 KB, що є найменшим показником.
3. PyCryptodome показала найнижчу продуктивність: час шифрування — 0.000123 секунд, а дешифрування — 0.000045 секунд. Споживання пам'яті становить 12.34 KB, що є середнім результатом.

Висновок:

- Якщо головною вимогою є швидкість, найкращим вибором є Cryptography.
- Для оптимізації споживання пам'яті слід використовувати OpenSSL.
- PyCryptodome підходить для завдань, де продуктивність не є критичною.

Таким чином, вибір бібліотеки залежить від конкретних потреб. Cryptography надає найшвидший результат, а OpenSSL є оптимальним компромісом між швидкістю та використанням пам'яті.