Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Фізико-технічний інститут

Методи реалізації криптографічних механізмів

Лабораторна робота №1

Виконала:
студентка ФІ-31мн
Панкєєва (Двоєглазова) С.Д.
Перевірила:
Селюх П. В.

Київ – 2024

Лабораторна робота №1. Вибір та реалізація базових фреймворків та бібліотек

Мета:

Вибір базових бібліотек/сервісів для подальшої реалізації криптосистеми.

Завдання:

Для другого типу лабораторних робіт — вибір бібліотеки реалізації основних криптографічних примітивів з точки зору їх ефективності за часом та пам'яттю для різних програмних платформ.

Порівняння бібліотек OpenSSL, Crypto++, CryptoLib, PyCrypto для розробки гібридної криптосистеми під Linux платформу.

Оформлення результатів роботи. Опис функції бібліотеки реалізації основних криптографічних примітивів обраної бібліотеки, з описом алгоритму, вхідних та вихідних даних, кодів повернення. Контрольний приклад роботи з функціями. Обґрунтування вибору бібліотеки.

Хід роботи:

Порівняння бібліотек

 OpenSSL

OpenSSL вважається золотим стандартом у світі криптографічного програмного забезпечення. Завдяки тому, що бібліотека написана на мові C, вона забезпечує високу продуктивність і підтримку широкого спектра алгоритмів, таких як AES, RSA, ECDSA та багато інших. OpenSSL активно використовується у промислових системах і серверних рішеннях. Однак її API може здатися складним, особливо для новачків, що може ускладнити процес розробки.

 Crypto++

Crypto++ — це сучасна бібліотека, яка написана на C++ і забезпечує кросплатформенну підтримку. Вона ідеально підходить для розробників, які працюють з об'єктно-орієнтованим підходом, оскільки має зручніший API порівняно з OpenSSL. Хоча продуктивність Crypto++ трохи поступається OpenSSL, її достатньо для більшості додатків, де важливі зручність та швидкість інтеграції.

 CryptoLib

CryptoLib орієнтована на мінімалізм і простоту. Вона часто використовується у навчальних проектах або вбудованих системах, де не потрібна велика кількість алгоритмів. Її головний недолік — обмежені можливості порівняно з більш повнофункціональними бібліотеками, такими як OpenSSL чи Crypto++.

 PyCrypto

PyCrypto створена спеціально для Python-розробників, що робить її дуже популярною серед розробників веб-додатків і невеликих проектів. Її API інтуїтивно зрозумілий, що дозволяє швидко реалізовувати криптографічні функції. Проте її продуктивність обмежена через реалізацію на інтерпретованій мові, що робить її менш придатною для задач, де потрібна висока швидкість обчислень.

1. **OpenSSL**

   Основні переваги:
   Висока продуктивність, підтримка великого спектра криптографічних алгоритмів, широка документація, підтримка багатьох платформ.

   Недоліки:
   Складність використання для новачків, великий обсяг коду.

2. **Crypto++**

   Основні переваги:
   Простота інтеграції з C++, модульність, доступність реалізацій новітніх алгоритмів.

   Недоліки:
   Менша продуктивність у порівнянні з OpenSSL, слабша підтримка.

3. **CryptoLib**

   Основні переваги:
   Легкість в освоєнні, орієнтована на початкові рішення.

   Недоліки:
   Низька продуктивність, обмежений набір алгоритмів.

4. **PyCrypto**

   Основні переваги:
   Інтуїтивно зрозумілий API, швидка інтеграція для Python-проектів.

   Недоліки:
   Застарілість, низька продуктивність у порівнянні з OpenSSL.

Вибір бібліотеки

Для реалізації гібридної криптосистеми під платформу Windows було обрано бібліотеку OpenSSL через її:

- високу продуктивність;
- підтримку широкого набору криптографічних алгоритмів;
- стабільність та перевірену надійність.

Опис програми :

- **`generate_aes_key`**

Генерує 256-бітний AES-ключ для симетричного шифрування.

- **`encrypt_with_aes`**

Шифрує передане повідомлення у режимі CFB, що забезпечує потокове шифрування. Повертає вектор ініціалізації та зашифрований текст.

- **`generate_rsa_keypair`**

Створює пару ключів RSA (2048 біт) для асиметричної криптографії.

- **`sign_data`**

Використовує приватний ключ для створення цифрового підпису з використанням алгоритму PSS і хеша SHA-256.

- **`verify_signature`**

Перевіряє достовірність підпису з використанням публічного ключа. У разі невідповідності повертає `False`.

- `data = 'Важливі дані'.encode('utf-8').`

Це конвертує текст у формат байтів за допомогою кодування UTF-8.

- У рядку виводу даних для підпису додано `data.decode("utf-8")`, щоб відобразити текст у вихідному вигляді.

Функції:

1. `generate_aes_key`

- Ця функція генерує випадковий 256-бітний ключ для AES-шифрування (Advanced Encryption Standard). Використовується генератор випадкових чисел, що забезпечує високий рівень криптографічної стійкості.
- **Вихідні дані:** 256-бітний AES-ключ у вигляді байтів (`bytes`).

2. `encrypt_with_aes`

- Функція шифрує текстове повідомлення за допомогою алгоритму AES у режимі CFB (Cipher Feedback Mode). Режим CFB дозволяє шифрувати дані потоково, що зручно для роботи з динамічними повідомленнями. Для шифрування генерується випадковий вектор ініціалізації (IV).
- **Вхідні дані:**
    - `key` (`bytes`): AES-ключ, згенерований функцією `generate_aes_key`.
    - `plaintext` (`str`): Текстове повідомлення, яке потрібно зашифрувати.
- **Вихідні дані:**
    - `iv` (`bytes`): Вектор ініціалізації (16 байтів).
    - `ciphertext` (`bytes`): Зашифроване повідомлення у вигляді байтів.

3. `generate_rsa_keypair`

- Ця функція генерує пару ключів RSA (приватний та публічний). Приватний ключ використовується для підпису даних, а публічний — для їхньої перевірки.
- **Вхідні дані:**
  Немає (функція самостійно генерує ключі).
- **Вихідні дані:**

    - `private_key` (`RSAPrivateKey`): Приватний ключ RSA.
    - `public_key` (`RSAPublicKey`): Публічний ключ RSA.

4. `sign_data`

- Функція створює цифровий підпис для переданих даних. Цей підпис генерується за допомогою приватного RSA-ключа. Підпис гарантує, що дані не були змінені та підтверджує автентичність автора.
- **Вхідні дані:**

    - `private_key` (`RSAPrivateKey`): Приватний ключ для створення підпису.
    - `data` (`bytes`): Дані, які потрібно підписати (наприклад, текст або файл у вигляді байтів).

- **Вихідні дані:**

- `signature` (`bytes`): Згенерований цифровий підпис для переданих даних.

5. `verify_signature`

- Функція перевіряє, чи є підпис дійсним. Для цього використовується публічний RSA-ключ. Якщо підпис відповідає даним, функція повертає `True`, інакше — `False`.
- **Вхідні дані:**
    - `public_key` (`RSAPublicKey`): Публічний ключ для перевірки підпису.
    - `signature` (`bytes`): Цифровий підпис, який потрібно перевірити.
    - `data` (`bytes`): Дані, для яких був створений підпис.
- **Вихідні дані:**
    - `True`, якщо підпис валідний.
    - `False`, якщо перевірка не вдалася.

6. `if __name__ == '__main__':`

Цей блок демонструє роботу всього коду:

Генерація AES-ключа та шифрування:

- Генерується AES-ключ.
- Шифрується текстове повідомлення ("Конфіденційна інформація").
- На виході — вектор ініціалізації та зашифроване повідомлення, які друкуються у консоль.

Генерація RSA-ключів:

- Генерується пара RSA-ключів (приватний і публічний).

Підпис даних:

- Дані " важливі дані" підписуються приватним ключем RSA.
- Підпис друкується у консоль.

Перевірка підпису:

- Публічний ключ використовується для перевірки підпису.

- Результат перевірки (`True` або `False`) також виводиться в консоль.

Таким чином, цей код демонструє основні операції криптографії: шифрування даних, створення цифрового підпису та перевірку його дійсності.

Код програми :

[2]:
```python
import os
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.asymmetric import padding, rsa
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.primitives.hashes import SHA256
```

[3]:
```python
def generate_aes_key() -> bytes:
    """Генерує випадковий ключ AES.

    Повертає:
        bytes: Згенерований 256-бітний AES-ключ.
    """
    return os.urandom(32)  # Генерується 32 байти випадкових даних (256 біт).


def encrypt_with_aes(key: bytes, plaintext: str) -> tuple[bytes, bytes]:
    """Шифрує текст за допомогою AES у режимі CFB.

    Аргументи:
        key (bytes): AES-ключ для шифрування.
        plaintext (str): Повідомлення, яке потрібно зашифрувати.

    Повертає:
        tuple[bytes, bytes]: Кортеж, що містить вектор ініціалізації (IV) та зашифрований текст.
    """
    iv = os.urandom(16)  # Генерується 16-байтний вектор ініціалізації.
    cipher = Cipher(algorithms.AES(key), modes.CFB(iv), backend=default_backend())
    encryptor = cipher.encryptor()
    ciphertext = encryptor.update(plaintext.encode('utf-8')) + encryptor.finalize()  # Шифрується повідомлення.

    return iv, ciphertext
```

[4]:
```python
def generate_rsa_keypair() -> tuple[rsa.RSAPrivateKey, rsa.RSAPublicKey]:
    """Генерує пару ключів RSA (приватний та публічний).

    Повертає:
        tuple[rsa.RSAPrivateKey, rsa.RSAPublicKey]: Кортеж із приватним та публічним RSA-ключами.
    """
    private_key = rsa.generate_private_key(public_exponent=65537, key_size=2048, backend=default_backend())
    public_key = private_key.public_key()

    return private_key, public_key


def sign_data(private_key: rsa.RSAPrivateKey, data: bytes) -> bytes:
    """Підписує дані за допомогою приватного ключа RSA.

    Аргументи:
        private_key (rsa.RSAPrivateKey): Приватний RSA-ключ для підпису.
        data (bytes): Дані для підпису.

    Повертає:
        bytes: Згенерований підпис.
    """
    return private_key.sign(
        data,
        padding.PSS(mgf=padding.MGF1(SHA256()), salt_length=padding.PSS.MAX_LENGTH),
        hashes.SHA256(),
    )
```

```python
[5]: def verify_signature(public_key: rsa.RSAPublicKey, signature: bytes, data: bytes) -> bool:
    """Перевіряє RSA-підпис.

    Аргументи:
        public_key (rsa.RSAPublicKey): Публічний ключ RSA для перевірки.
        signature (bytes): Підпис для перевірки.
        data (bytes): Оригінальні дані.

    Повертає:
        bool: True, якщо підпис валідний, False у разі помилки.
    """
    try:
        public_key.verify(
            signature,
            data,
            padding.PSS(mgf=padding.MGF1(SHA256()), salt_length=padding.PSS.MAX_LENGTH),
            hashes.SHA256(),
        )
        return True
    except Exception:
        return False  # Якщо перевірка не пройшла, повертається False.
```

```python
if __name__ == '__main__':
    # Демонстрація шифрування AES
    original_message = 'Конфіденційна інформація'
    aes_key = generate_aes_key()
    iv, ciphertext = encrypt_with_aes(aes_key, original_message)

    print("Лабораторна робота 1 , Панкєєва(Двоєглазова) СД")
    print(f'Оригінальне повідомлення: {original_message}')
    print(f'AES-ключ: {aes_key.hex()}')
    print(f'Вектор ініціалізації (IV): {iv.hex()}')
    print(f'Зашифрований текст: {ciphertext.hex()}')

    # Генерація RSA-ключів та підпис
    private_key, public_key = generate_rsa_keypair()
    data = 'Важливі дані'.encode('utf-8')  # Текст перекодується в байти за допомогою UTF-8
    signature = sign_data(private_key, data)

    print(f'Дані для підпису: {data.decode("utf-8")}')
    print(f'Підпис: {signature.hex()}')

    # Перевірка підпису
    is_valid = verify_signature(public_key, signature, data)
    print(f'Підпис валідний: {is_valid}')
```

Скрін виконання програми :

```
Лабораторна робота 1 , Панкєєва(Двоєглазова) СД
Оригінальне повідомлення: Конфіденційна інформація
AES-ключ: 74d9f46f9c145d73f22fe70c04d838d98d2e8c17e52649e4baac0b558ab739b9
Вектор ініціалізації (IV): f25922bd3a421a669def5388625fb2de
Зашифрований текст: 34513c38388969c9a699f8dd44fbd6b14c3204ef6202ff718a67fc90bf0f4bf17e9eead986030b2e94b04d3d80f716
Дані для підпису: Важливі дані
Підпис: 51749a8148042408360acb949e6cb26621455f0f64a34b1f177d03454bfab3c972973d2dfafbc227519ba03381a16b835715015f79e17f11ca40a3a4463c3ca71a3cf2bbdcb0dd2e7
d4f2f695d3a23fd99abb90bc4ba8e26786931229f1d99c6f80d656977b81e1bf54100e6f7bacac3cae8001f4463cb0a6f1907e916d74f6ce499169ee668bd86d90ad3640120cdb5c013c5ddc5
c51fc27f6ad1d96dbbba020efe0664c18a486af7b8737f102494fc043d8667cd9689e5e0589c6baffaed04ec8a35fe275cef5eb0793b18d50c83488ae88b7d0219b91253c4997d2de2605f14b
d3b1a029fc8f4b30590d1ef37b901c6335e681e49b3baa2036114fa1a1221
Підпис валідний: True
```

Використана література :

1. Задірака В.К., Олексюк О.С. Комп’ютерна арифметика багаторозряних чисел: Наукове видання. – Київ: 2003. – 264 с.
2. Д. Э. Кнут Искусство программирования, том 2. Получисленные алгоритмы, 3-е изд.: Пер. с англ.: Уч. пос.–М: Изд. дом «Вильямс», 2001.–832 с.
3. C.K.Koc High-Speed RSA Implementation TR RSA Data Security