

Лаб 1

Варіант 2В 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.

Зміст

OpenSSL.....	2
Алгоритм.....	2
Код.....	3
Результат.....	4
crypto++.....	6
Алгоритм.....	6
Код.....	7
Результат.....	9
CryptoLib.....	11
PyCrypto.....	12
Алгоритм.....	12
Код.....	13
Результат.....	14
Порівняння бібліотек.....	16
Критерії порівняння.....	16
Вхідні дані.....	16
Візуалізація результатів.....	17
Висновки.....	18

Гібридна криптосистема, означає, що вона викроситовує асиметричне шифрування RSA та симетричне шифрування AES

OpenSSL

Дуже потужна бібліотека. Для багатьох лінукс систем її можна встановити через apt get install, або вона може бути предустановлена, якщо це kali linux. Рекомендується використовувати саме команди openssl через термінал, чим робити c++ код.

Алгоритм

Генерація закритого ключа RSA (2048 біт):

Вхідні дані: Відсутні.

Вихідні дані: Закритий ключ RSA зберігається в файлі private_key.pem.

Код повернення: 0 (успішно).

Вилучення публічного ключа з закритого ключа RSA:

Вхідні дані: Закритий ключ RSA з файлу private_key.pem.

Вихідні дані: Публічний ключ RSA зберігається в файлі public_key.pem.

Код повернення: 0 (успішно).

Генерація випадкового симетричного ключа AES:

Вхідні дані: Відсутні.

Вихідні дані: Випадковий симетричний ключ AES розміром 32 байти зберігається в файлі symmetric_key.bin.

Код повернення: 0 (успішно).

Шифрування даних симетричним ключем (AES):

Вхідні дані: Дані з файлу plaintext.txt, симетричний ключ з файлу symmetric_key.bin.

Вихідні дані: Зашифровані дані зберігаються у файлі encrypted_data.enc.

Код повернення: 0 (успішно).

Шифрування симетричного ключа публічним ключем RSA:

Вхідні дані: Публічний ключ RSA з файлу public_key.pem, симетричний ключ з файлу symmetric_key.bin.

Вихідні дані: Зашифрований симетричний ключ зберігається в файлі encrypted_symmetric_key.bin.

Код повернення: 0 (успішно).

Розшифрування симетричного ключа за допомогою закритого ключа RSA:

Вхідні дані: Закритий ключ RSA з файлу private_key.pem, зашифрований симетричний ключ з файлу encrypted_symmetric_key.bin.

Вихідні дані: Розшифрований симетричний ключ зберігається в файлі symmetric_key.bin.

Код повернення: 0 (успішно).

Розшифрування даних симетричним ключем (AES):

Вхідні дані: Зашифровані дані з файлу encrypted_data.enc, симетричний ключ з файлу symmetric_key.bin.

Вихідні дані: Розшифровані дані зберігаються в файлі decrypted_data.txt.

Код повернення: 0 (успішно).

Вивід результатів:

Вхідні дані: Зашифрований ключ RSA, розшифрований ключ AES, зашифрований текст, розшифрований текст.

Вихідні дані: Вивід в консолі скрипта.

Код повернення: 0 (успішно).

Код

Openssl.sh

#!/bin/bash

start_time=\$(date +%s.%N) # Запуск таймера

Генерация закрытого ключа RSA (2048 бит)

openssl genpkey -algorithm RSA -out private_key.pem

echo "Закрытый ключ RSA сгенерирован."

Извлечение публичного ключа из закрытого ключа

openssl rsa -pubout -in private_key.pem -out public_key.pem

echo "Публичный ключ RSA извлечен."

Генерация случайного симметричного ключа AES

openssl rand -out symmetric_key.bin 32

echo "Симметричный ключ AES сгенерирован."

Шифрование данных с использованием симметричного ключа (AES)

openssl enc -aes-256-cfb -in plaintext.txt -out encrypted_data.enc -pass file:symmetric_key.bin -pbkdf2

echo "Данные успешно зашифрованы с использованием симметричного ключа AES."

Шифрование симметричного ключа с использованием публичного ключа RSA

openssl pkeyutl -encrypt -pubin -inkey public_key.pem -in symmetric_key.bin -out encrypted_symmetric_key.bin

echo "Симметричный ключ успешно зашифрован с использованием публичного ключа RSA."

Расшифрование симметричного ключа с использованием закрытого ключа RSA

openssl pkeyutl -decrypt -inkey private_key.pem -in encrypted_symmetric_key.bin -out symmetric_key.bin

echo "Симметричный ключ успешно расшифрован с использованием закрытого ключа RSA."

Расшифрование данных симметричным ключом (AES)

openssl enc -d -aes-256-cfb -in encrypted_data.enc -out decrypted_data.txt -pass file:symmetric_key.bin -pbkdf2

echo "Данные успешно расшифрованы с использованием симметричного ключа AES."

Вывод результатов

echo "Зашифрованный ключ RSA:"

cat encrypted_symmetric_key.bin

```

echo "Расшифрованный ключ AES:"
cat symmetric_key.bin
echo "Зашифрованный текст:"
cat encrypted_data.enc
echo "Расшифрованный текст:"
cat decrypted_data.txt
end_time=$(date +%s.%N) # Остановка таймера
execution_time=$(echo "$end_time - $start_time" | bc)

echo "Время выполнения скрипта: $execution_time секунд"

```

Результат

```

(kali@kali) ~/lab1/openssl
$ ./openssl.sh
.....
Закрытый ключ RSA сгенерирован.
writing RSA key
Публичный ключ RSA извлечен.
Симметричный ключ AES сгенерирован.
Данные успешно зашифрованы с использованием симметричного ключа AES.
Симметричный ключ успешно зашифрован с использованием публичного ключа RSA.
Симметричный ключ успешно расшифрован с использованием закрытого ключа RSA.
Данные успешно расшифрованы с использованием симметричного ключа AES.
Зашифрованный ключ RSA:
*****[Q*****<*****6A*****-***yB54**/!-***l**/*]*****bX**d**[eD]*****BEt*****Pe*****9Ct-s0zN**dc*****vL-$p1GVm**Расшифрованный ключ AES:
**H Зашифрованный текст:
Salted__Bu*****E0M4G
Расшифрованный текст:
Oleksii
Время выполнения скрипта: .275176602 секунд

```

Тепер перевіримо час та пам'ять. Хоча в коді і указан час, однак краще його додатково перевірити через команду time, бо ця команда буде використана і для інших бібліотек

```

(kali@kali) ~/lab1/openssl
$ time ./openssl.sh
.....
Закрытый ключ RSA сгенерирован.
writing RSA key
Публичный ключ RSA извлечен.
Симметричный ключ AES сгенерирован.
Данные успешно зашифрованы с использованием симметричного ключа AES.
Симметричный ключ успешно зашифрован с использованием публичного ключа RSA.
Симметричный ключ успешно расшифрован с использованием закрытого ключа RSA.
Данные успешно расшифрованы с использованием симметричного ключа AES.
Зашифрованный ключ RSA:
*****[Q*****<*****6A*****-***yB54**/!-***l**/*]*****bX**d**[eD]*****BEt*****Pe*****9Ct-s0zN**dc*****vL-$p1GVm**Расшифрованный ключ AES:
**H Зашифрованный текст:
Salted__Bu*****E0M4G
Расшифрованный текст:
Oleksii
Время выполнения скрипта: .769373166 секунд

real    0.78s
user    0.55s
sys     0.20s
cpu     96%

```

Знадобилося 0.78 секунд для реалізації алгоритма

Бачимо результат пам'яті

```

        ♦Зашифрованный текст:
Salted_♦♦♦♦!♦9♦♦иРасшифрованный текст:
Oleksii
=65210=
=65210= HEAP SUMMARY:
=65210=     in use at exit: 69,292 bytes in 848 blocks
=65210=     total heap usage: 2,966 allocs, 2,118 frees, 133,069 bytes allocated
=65210=
=65210= LEAK SUMMARY:
=65210=     definitely lost: 0 bytes in 0 blocks
=65210=     indirectly lost: 0 bytes in 0 blocks
=65210=     possibly lost: 0 bytes in 0 blocks
=65210=     still reachable: 69,292 bytes in 848 blocks
=65210=     suppressed: 0 bytes in 0 blocks
=65210= Reachable blocks (those to which a pointer was found) are not shown.
=65210= To see them, rerun with: --leak-check=full --show-leak-kinds=all
=65210=
=65210= For lists of detected and suppressed errors, rerun with: -s
=65210= ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
=65209=
=65209= HEAP SUMMARY:
=65209=     in use at exit: 68,834 bytes in 831 blocks
=65209=     total heap usage: 2,959 allocs, 2,128 frees, 132,527 bytes allocated
=65209=
=65209= LEAK SUMMARY:
=65209=     definitely lost: 0 bytes in 0 blocks
=65209=     indirectly lost: 0 bytes in 0 blocks
=65209=     possibly lost: 0 bytes in 0 blocks
=65209=     still reachable: 68,834 bytes in 831 blocks
=65209=     suppressed: 0 bytes in 0 blocks
=65209= Reachable blocks (those to which a pointer was found) are not shown.
=65209= To see them, rerun with: --leak-check=full --show-leak-kinds=all
=65209=
=65209= For lists of detected and suppressed errors, rerun with: -s
=65209= ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

```

crypto++

Ця бібліотека може бути використана тільки на мові c++ . Для лінукс її можна встановити за доп. `sudo apt-get install libcrypto++-dev`

Взагалі, робота з бібліотеками c++ на Лінукс не є дуже гарною ідеєю, оскільки вимагає встановлення компілятора та роботою з ним.

Алгоритм

Генерація закритого ключа RSA (2048 біт):

Вхідні дані: Відсутні.

Вихідні дані: Закритий ключ RSA зберігається у файлі `private_key.pem`.

Код повернення: 0 (успішно).

Отримання публічного ключа з закритого ключа RSA:

Вхідні дані: Закритий ключ RSA з файлу `private_key.pem`.

Вихідні дані: Публічний ключ RSA зберігається у файлі `public_key.pem`.

Код повернення: 0 (успішно).

Генерація випадкового симетричного ключа AES з обраною довжиною (32 байти в даному випадку).

Вхідні дані: Відсутні.

Вихідні дані: Випадковий симетричний ключ AES зберігається в файлі `symmetric_key.bin`.

Код повернення: 0 (успішно).

Шифрування даних з використанням симетричного ключа AES з режимом CBC (Cipher Block Chaining).

Вхідні дані: Текст для шифрування та симетричний ключ AES.

Вихідні дані: Зашифровані дані зберігаються в змінній `ciphertext`.

Код повернення: 0 (успішно).

Шифрування симетричного ключа AES з використанням публічного ключа RSA.

Вхідні дані: Симетричний ключ AES та публічний ключ RSA.

Вихідні дані: Зашифрований симетричний ключ зберігається в змінній `encryptedSymmetricKey`.

Код повернення: 0 (успішно).

Розшифрування симетричного ключа AES з використанням закритого ключа RSA.

Вхідні дані: Зашифрований симетричний ключ та закритий ключ RSA.

Вихідні дані: Розшифрований симетричний ключ зберігається в змінній `decryptedSymmetricKey`.

Код повернення: 0 (успішно).

Розшифрування зашифрованих даних з використанням симетричного ключа AES та режиму CBC.

Вхідні дані: Зашифрований текст та симетричний ключ AES.

Вихідні дані: Розшифрований текст зберігається в змінній recoveredPlaintext.

Код повернення: 0 (успішно).

Вивід результатів:

Виведення зашифрованого ключа RSA, розшифрованого ключа AES, зашифрованого тексту та розшифрованого тексту в консолі.

Вхідні дані: Результати попередніх операцій.

Вихідні дані: Вивід в консолі.

Код повернення: 0 (успішно).

Код

```
#include <iostream>
#include <fstream>
#include <cryptopp/aes.h>
#include <cryptopp/rsa.h>
#include <cryptopp/osrng.h>
#include <cryptopp/files.h>
#include <cryptopp/modes.h>

int main() {
    CryptoPP::AutoSeededRandomPool rng;

    // Генерація закритого ключа RSA (2048 бит)
    CryptoPP::RSA::PrivateKey privateKey;
    privateKey.GenerateRandomWithKeySize(rng, 2048);

    // Сохранение закрытого ключа в файл
    CryptoPP::FileSink privateKeyFile("private_key.pem");
    privateKey.Save(privateKeyFile);

    std::cout << "Закрытый ключ RSA сгенерирован." << std::endl;

    // Получение публичного ключа из закрытого ключа
    CryptoPP::RSA::PublicKey publicKey;
    publicKey = privateKey;

    // Сохранение публичного ключа в файл
    CryptoPP::FileSink publicKeyFile("public_key.pem");
    publicKey.Save(publicKeyFile);

    std::cout << "Публичный ключ RSA извлечен." << std::endl;

    // Генерація випадкового симетричного ключа AES
    CryptoPP::SecByteBlock symmetricKey(32); // Выберите длину ключа AES
    rng.GenerateBlock(symmetricKey, symmetricKey.size());
```

```

// Сохранение симметричного ключа в файл
std::ofstream symmetricKeyFile("symmetric_key.bin", std::ios::binary);
symmetricKeyFile.write(reinterpret_cast<char*>(symmetricKey.BytePtr()),
symmetricKey.size());
symmetricKeyFile.close();

std::cout << "Симметричный ключ AES сгенерирован." << std::endl;

// Загрузка симметричного ключа из файла
std::ifstream symmetricKeyFileIn("symmetric_key.bin", std::ios::binary);
symmetricKeyFileIn.read(reinterpret_cast<char*>(symmetricKey.BytePtr()),
symmetricKey.size());
symmetricKeyFileIn.close();

// Шифрование данных с использованием симметричного ключа (AES)
std::string plaintext = "Oleksii";
std::string ciphertext;

CryptoPP::byte iv[CryptoPP::AES::BLOCKSIZE];
rng.GenerateBlock(iv, sizeof(iv));

CryptoPP::AES::Encryption aesEncryption(symmetricKey.BytePtr(), symmetricKey.size());
CryptoPP::CBC_Mode_ExternalCipher::Encryption cbcEncryption(aesEncryption, iv);

CryptoPP::StreamTransformationFilter encryptor(cbcEncryption, new
CryptoPP::StringSink(ciphertext));
encryptor.Put(reinterpret_cast<const unsigned char*>(plaintext.data()), plaintext.size());
encryptor.MessageEnd();

std::cout << "Данные успешно зашифрованы с использованием симметричного ключа
AES." << std::endl;

// Шифрование симметричного ключа с использованием публичного ключа RSA
CryptoPP::RSAES_OAEP_SHA_Encryptor rsaEncryptor(publicKey);
std::string encryptedSymmetricKey;

CryptoPP::StringSource s(symmetricKey.BytePtr(), symmetricKey.size(), true, new
CryptoPP::PK_EncryptorFilter(rng, rsaEncryptor, new
CryptoPP::StringSink(encryptedSymmetricKey)));

std::cout << "Симметричный ключ успешно зашифрован с использованием публичного
ключа RSA." << std::endl;

// Расшифрование симметричного ключа с использованием закрытого ключа RSA
CryptoPP::RSAES_OAEP_SHA_Decryptor rsaDecryptor(privateKey);
std::string decryptedSymmetricKey;

CryptoPP::StringSource ss(encryptedSymmetricKey, true, new
CryptoPP::PK_DecryptorFilter(rng, rsaDecryptor, new
CryptoPP::StringSink(decryptedSymmetricKey)));

```



```

std::cout << "Симметричный ключ успешно расшифрован с использованием закрытого
ключа RSA." << std::endl;

// Расшифрование данных симметричным ключом (AES)
std::string recoveredPlaintext;

CryptoPP::AES::Decryption aesDecryption(symmetricalKey.BytePtr(), symmetricalKey.size());
CryptoPP::CBC_Mode_ExternalCipher::Decryption cbcDecryption(aesDecryption, iv);

CryptoPP::StreamTransformationFilter decryptor(cbcDecryption, new
CryptoPP::StringSink(recoveredPlaintext));
decryptor.Put(reinterpret_cast<const unsigned char*>(ciphertext.data()), ciphertext.size());
decryptor.MessageEnd();

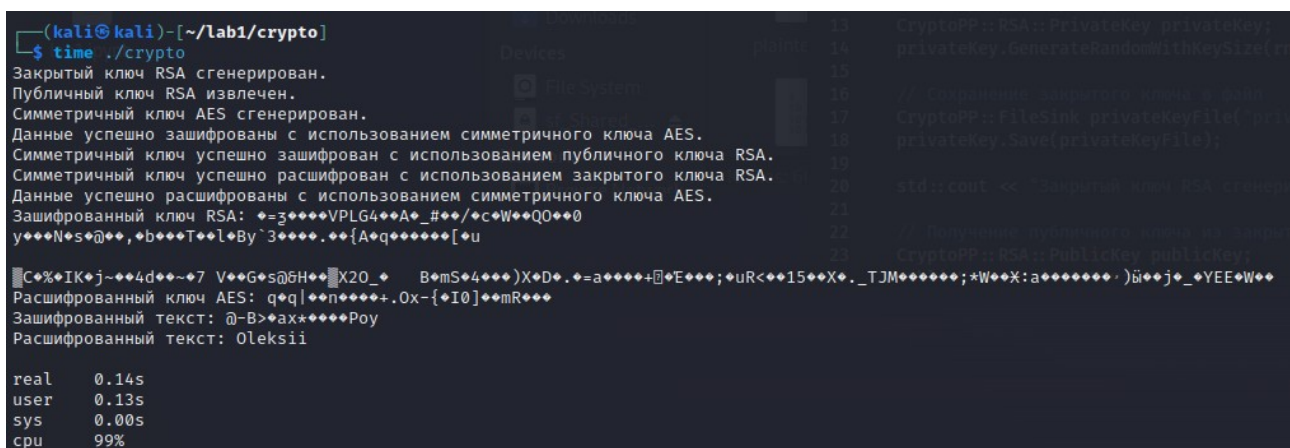
std::cout << "Данные успешно расшифрованы с использованием симметричного ключа
AES." << std::endl;

// Вывод результатов
std::cout << "Зашифрованный ключ RSA: " << encryptedSymmetricalKey << std::endl;
std::cout << "Расшифрованный ключ AES: " << decryptedSymmetricalKey << std::endl;
std::cout << "Зашифрованный текст: " << ciphertext << std::endl;
std::cout << "Расшифрованный текст: " << recoveredPlaintext << std::endl;

return 0;
}

```

Результат



```

(kali@kali)-[~/lab1/crypto]
$ time ./crypto
Закрытый ключ RSA сгенерирован.
Публичный ключ RSA извлечен.
Симметричный ключ AES сгенерирован.
Данные успешно зашифрованы с использованием симметричного ключа AES.
Симметричный ключ успешно зашифрован с использованием публичного ключа RSA.
Симметричный ключ успешно расшифрован с использованием закрытого ключа RSA.
Данные успешно расшифрованы с использованием симметричного ключа AES.
Зашифрованный ключ RSA: 3VPLG4A_#sWQ000
yN@s@,bTlBy`3{Aq*****[u
[С%IKj~4d~7 V+Gs@6H[X20_ BmS4****)X*D. =a****+E***;uR<15X*_TJM*****;*W*Ж:a***** )йj*_YEEW*
Расшифрованный ключ AES: q|n*****.0x-{I0]mR***
Зашифрованный текст: @-B>*ax*****Poу
Расшифрованный текст: Oleksii

real    0.14s
user    0.13s
sys     0.00s
cpu     99%

```

```
[kali@kali:~/lab1/crypto]
$ valgrind --tool=memcheck --leak-check-full ./crypto

==72227== Memcheck, a memory error detector
==72227== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==72227== Using Valgrind-3.19.0 and LibVEX; rerun with -h for copyright info
==72227== Command: ./crypto
==72227==
Закрытый ключ RSA сгенерирован.
Публичный ключ RSA извлечен.
Симметричный ключ AES сгенерирован.
Данные успешно зашифрованы с использованием симметричного ключа AES.
Симметричный ключ успешно зашифрован с использованием публичного ключа RSA.
Симметричный ключ успешно расшифрован с использованием закрытого ключа RSA.
Данные успешно расшифрованы с использованием симметричного ключа AES.
Зашифрованный ключ RSA: ==1 k==[#####]#0x####==h[

<=>i;[#####dml;1lw=====g[==q[e][e][.][_][_][_][_]#####qdc[e]kA##B###czBgde

eehb[iekceb+ec[vz2be"*S*duq+em+p

xx?o;I,[#####)*Ca **luc+c+[+-

***
+iEku

+<e>3*
Расшифрованный ключ AES: #####-0 ***-T#=====[2]
Зашифрованный текст: [E+<vYJW* *m
Расшифрованный текст: 0loksii

==72227==
==72227== HEAP SUMMARY:
==72227==   in use at exit: 88 bytes in 3 blocks
==72227== total heap usage: 8,140 allocs, 8,137 frees, 1,100,100 bytes allocated
==72227==
==72227== LEAK SUMMARY:
==72227==    definitely lost: 0 bytes in 0 blocks
==72227==    indirectly lost: 0 bytes in 0 blocks
==72227==    possibly lost: 0 bytes in 0 blocks
==72227==    still reachable: 88 bytes in 3 blocks
==72227==    suppressed: 0 bytes in 0 blocks
==72227== Reachable blocks (those to which a pointer was found) are not shown.
==72227== To see them, rerun with: --leak-check-full --show-leak-kinds=all
==72227==
==72227== For lists of detected and suppressed errors, rerun with: -s
==72227== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

CryptoLib

Дана бібліотека знаходиться у закритому доступі і тому неможливо її протестувати. Тому вона була вилучена з порівняння



PyCrypto

Ця бібліотека використовує python. Потрібно встановити цю бібліотеку через `pip install pycryptodome`

Алгоритм

Генерація закритого ключа RSA розміром 2048 біт.

Вхідні дані: Відсутні.

Вихідні дані: Закритий ключ RSA зберігається у файлі `private_key.pem`.

Код повернення: 0 (успішно).

Отримання публічного ключа з закритого ключа RSA.

Вхідні дані: Закритий ключ RSA.

Вихідні дані: Публічний ключ RSA зберігається у файлі `public_key.pem`.

Код повернення: 0 (успішно).

Генерація випадкового симетричного ключа AES з обраною довжиною (16 байт для AES-128).

Вхідні дані: Відсутні.

Вихідні дані: Випадковий симетричний ключ AES зберігається в файлі `symmetric_key.bin`.

Код повернення: 0 (успішно).

Шифрування даних з використанням симетричного ключа AES в режимі CBC (Cipher Block Chaining).

Вхідні дані: Текст для шифрування та симетричний ключ AES.

Вихідні дані: Зашифрований текст зберігається в змінній `ciphertext`.

Код повернення: Відсутній.

Шифрування симетричного ключа AES з використанням публічного ключа RSA з доповненням PKCS1 OAEP.

Вхідні дані: Симетричний ключ AES та публічний ключ RSA.

Вихідні дані: Зашифрований симетричний ключ зберігається в файлі `encrypted_symmetric_key.bin`.

Код повернення: Відсутній.

Розшифрування симетричного ключа AES з використанням закритого ключа RSA з доповненням PKCS1 OAEP.

Вхідні дані: Зашифрований симетричний ключ та закритий ключ RSA.

Вихідні дані: Розшифрований симетричний ключ зберігається в файлі `decrypted_symmetric_key.bin`.

Код повернення: Відсутній.

Розшифрування зашифрованих даних з використанням симетричного ключа AES в режимі CBC.

Вхідні дані: Зашифрований текст та розшифрований симетричний ключ.

Вихідні дані: Розшифрований текст.

Код повернення: Відсутній.

Код

```
#!/usr/bin/env python

from Crypto.Cipher import AES, PKCS1_OAEP
from Crypto.PublicKey import RSA
from Crypto.Random import get_random_bytes
from Crypto.Util.Padding import pad, unpad

# Генерація закритого ключа RSA (2048 бит)
private_key = RSA.generate(2048)

# Сохранение закритого ключа в файл
private_key_file = open("private_key.pem", "wb")
private_key_file.write(private_key.export_key())
private_key_file.close()

print("Закритий ключ RSA сгенерирован.")

# Получение публичного ключа из закритого ключа
public_key = private_key.publickey()

# Сохранение публичного ключа в файл
public_key_file = open("public_key.pem", "wb")
public_key_file.write(public_key.export_key())
public_key_file.close()

print("Публичный ключ RSA извлечен.")

# Генерація випадкового симетричного ключа AES
symmetric_key = get_random_bytes(16) # 16 байт для AES-128

# Сохранение симетричного ключа в файл
symmetric_key_file = open("symmetric_key.bin", "wb")
symmetric_key_file.write(symmetric_key)
symmetric_key_file.close()

print("Симметричный ключ AES сгенерирован.")

# Загрузка симетричного ключа из файла
symmetric_key_file_in = open("symmetric_key.bin", "rb")
symmetric_key = symmetric_key_file_in.read()
symmetric_key_file_in.close()
```

```

# Шифрование данных с использованием симметричного ключа (AES)
plaintext = "Oleksii"
iv = get_random_bytes(16) # Генерация случайного вектора инициализации
cipher = AES.new(symmetric_key, AES.MODE_CBC, iv)

# Добавляем дополнение PKCS7
padded_plaintext = pad(plaintext.encode('utf-8'), AES.block_size)
ciphertext = iv + cipher.encrypt(padded_plaintext)

# Шифрование симметричного ключа с использованием публичного ключа RSA
rsa_cipher = PKCS1_OAEP.new(public_key)
encrypted_symmetric_key = rsa_cipher.encrypt(symmetric_key)

encrypted_symmetric_key_file = open("encrypted_symmetric_key.bin", "wb")
encrypted_symmetric_key_file.write(encrypted_symmetric_key)
encrypted_symmetric_key_file.close()
print("Симметричный ключ успешно зашифрован с использованием публичного ключа RSA.")

# Расшифрование симметричного ключа с использованием закрытого ключа RSA
rsa_decipher = PKCS1_OAEP.new(private_key)
decrypted_symmetric_key = rsa_decipher.decrypt(encrypted_symmetric_key)

decrypted_symmetric_key_file = open("decrypted_symmetric_key.bin", "wb")
decrypted_symmetric_key_file.write(decrypted_symmetric_key)
decrypted_symmetric_key_file.close()
print("Симметричный ключ успешно расшифрован с использованием закрытого ключа RSA.")

# Расшифрование данных симметричным ключом (AES)
iv = ciphertext[:16]
cipher = AES.new(decrypted_symmetric_key, AES.MODE_CBC, iv)
padded_plaintext = cipher.decrypt(ciphertext[16:])
plaintext = unpad(padded_plaintext, AES.block_size).decode('utf-8')

# Вывод зашифрованного ключа RSA
print("Зашифрованный ключ RSA:", encrypted_symmetric_key.hex())

# Вывод расшифрованного ключа AES
print("Расшифрованный ключ AES:", decrypted_symmetric_key.hex())

# Вывод зашифрованного текста
print("Зашифрованный текст:", ciphertext.hex())
print("Данные успешно зашифрованы с использованием симметричного ключа AES.")

print("Данные успешно расшифрованы с использованием симметричного ключа AES.")
print("Расшифрованный текст:", plaintext)

```

Результат

```
[kali@kali]~/lab1/pycrypto$ ./crypto.py
Закрытый ключ RSA сгенерирован.
Публичный ключ RSA извлечен.
Симметричный ключ AES сгенерирован.
Симметричный ключ успешно зашифрован с использованием публичного ключа RSA.
Симметричный ключ успешно расшифрован с использованием закрытого ключа RSA.
Зашифрованный ключ RSA: 819b68689e148b5f816dbf650b11ad51081da11e891c9c370b0806f3bcddf474b31eece639ed22b538d673b338b0da7e755358ac18031b2bcfd4d5287ee965eedd0985952df849081baa097ec81d6827f514d2c3256ee3a7385b662ca487b085d4fd418fd496b14c
f87c452c0b0449a1501ff3b5a7edcd0734ee8315c39a90a07df3040b58870211e52c0844e25cfbc82fd2c3c614f0444ed369f0931982cab50ff18402ef742cf0262da3162a2a9e01b7eb4f0d730579173c77cb417deedc581a9fa3bdc9c9cd6b57366791511e59986e94d1750df8b9c58a226aabc
f36516d138c03c096e7a2a550d49cb7db28c196a16822a2c346420db286039fb
Расшифрованный ключ AES: e90fc2773ba1990eb3eafa07514a7270
Зашифрованный текст: da1e9c3c270c3d61a4cf50d4ac3b1370909f4895c7638a751c13bf76909fb6
Данные успешно зашифрованы с использованием симметричного ключа AES.
Данные успешно расшифрованы с использованием симметричного ключа AES.
Расшифрованный текст: Olexs11

real    0.52s
user    0.51s
sys      0.02s
cpu      99%
```

```
[kali@kali]~/lab1/pycrypto$ valgrind --tool=memcheck --leak-check=full ./crypto.py
==84708== Memcheck, a memory error detector
==84708== Copyright (c) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==84708== Using Valgrind-3.19.0 and LibVEX; rerun with -h for copyright info
==84708== Command: ./crypto.py
==84708==
Закрытый ключ RSA сгенерирован.
Публичный ключ RSA извлечен.
Симметричный ключ AES сгенерирован.
Симметричный ключ успешно зашифрован с использованием публичного ключа RSA.
Симметричный ключ успешно расшифрован с использованием закрытого ключа RSA.
Зашифрованный ключ RSA: 35873f1bf44k8098aa3b2205c2ec6aaad7f7eb5aa360b3d62cab2aa7a3e7b132791464c4081d70868008956558cdddb8b7e33772c3a579f1ffccdc0437d05cd916b35ff4439cb57399cd07bd45dc065d037ee815295155a3e61d7b0f98fb26f3975216ba8db2
a028701c0bc130ac70a8f1122c0b6eb589508e5c727146fcs096657b0f0a9918f523f1b809c42ff4fbc894ef883db2c90ce08c91d0d6a7223d504nacd19d3bfad55642854b8a1007cbcd79ecd5d595461ee8e2a5544ee082c0030b0e07260a779226dc228e17c1cb114f626a3ce0754b1d6ef8b542
25c6abf3384af170d99f3c19f0cb8d419d3e49774ee5b5035a126aa16936ac31
Расшифрованный ключ AES: c4085eakfa2344fe080c4af0b09c3a4f
Зашифрованный текст: 30d1a0f92e86a4f65120ebcecf085111f81b1c1ccddc77486dc36677722ea234
Данные успешно зашифрованы с использованием симметричного ключа AES.
Данные успешно расшифрованы с использованием симметричного ключа AES.
Расшифрованный текст: Olexs11
```

Вітіку пам’яті немає

Порівняння бібліотек

Критерії порівняння

1. Час — характеризує час, який знадобився для реалізації алгоритму, менший результат - краще
2. Витік пам'яті — характеризує використання пам'яті та її витік, менший результат - краще
3. Мова програмування — характеризує, яка мова програмування була використана для створення алгоритму
4. Складність реалізації — характеризує рівень складності реалізації, менший результат - краще
5. Встановлення за замовченням — характеризує чи були необхідні бібліотеки встановленні в kali linux за замовченням

Вхідні дані

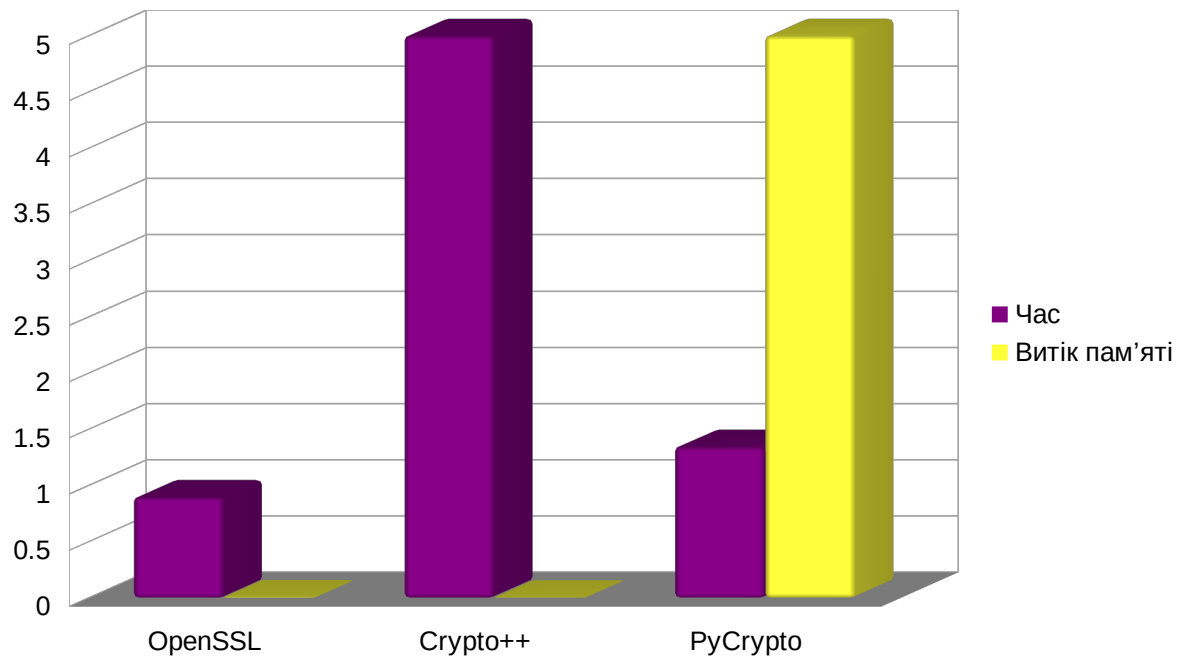
	Час (секунд)	Витік пам'яті (байтів)	Мова програмування	Складність реалізації	Встановлення за замовченням
OpenSSL	0.78	69.292	Bash / c++	2	5
Crypto++	0.14	88	c++	5	3
PyCrypto	0.52	0	python	4	3

Переведення в 5-ти бальну шкалу

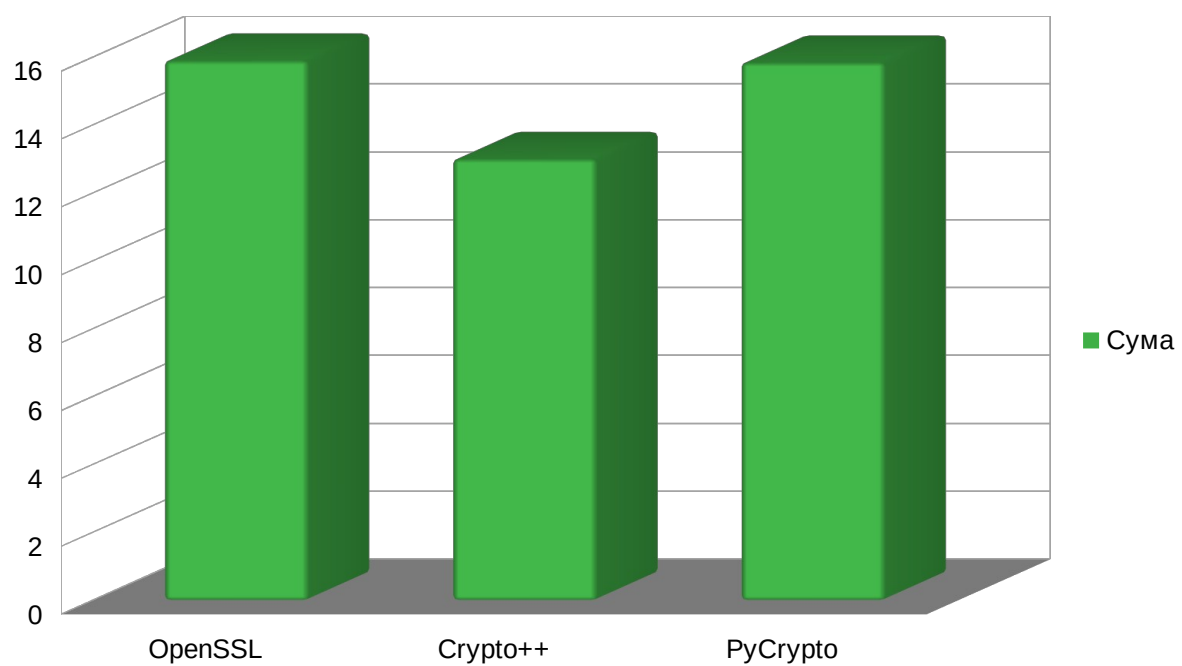
	Час	Витік пам'яті	Мова програмування	Складність реалізації	Встановлення за замовченням	Сума
OpenSSL	0.897	0.007	5	5	5	15.905
Crypto++	5	0.006	3	2	3	13.006
PyCrypto	1.346	5	4	2.5	3	15.846

Візуалізація результатів

Порівняння результатів порівняння витіку пам'яті та часу роботи алгоритму. Більший бал - краще



Порівнянн загальних результатів



Висновки

В результаті проведення дослідження було визначено:

1. Crypto++ може запропонувати найбільшу швидкість, ймовірно через використання C++, але має найбільші витрати пам'яті, знову ж таки, ймовірно через використання C++. Рекомендується використовувати Crypto++, у задачах, коли потрібно досягти найбільшу швидкість виконання, при умові, що фахівець обізнаний з C++. Це є важливий пункт, оскільки реалізації гібридної криптосистеми на Crypto++ є досить непростю задачею.
2. Pycrypto, який використовує python, показав найкращий результат в проблемі витрати пам'яті. В нього її немає. Тому у задачах де рекомендується мінімальний витік пам'яті, або його відсутність, рекомендується використовувати Pycrypto. Через використання мови програмування python, робить його реалізацію простішим аніж Crypto++
3. OpenSSL, показав найгірший результат в швидкості виконання алгоритму і показав присутність витіку пам'яті. Проте він є встановлений за замовченням в kali linux, та є наймовірно простим в реалізації гібридної криптосистеми. Рекомендується його використовувати новачкам, оскільки він потребує мінімальне знання програмування, скоріше знання користування утилітою openssl або користувачам яким не важливий витік пам'яті або швидкість виконання алгоритму.
4. Для Linux систем бажано використовувати або OpenSSL, або Pycrypto, оскільки в першому випадку, воно саме заточена під Linux, бо його можна встановити та використовувати через термінал утиліту, а Pycrypto, бо для Linux мова програмування Python є більш розповсюдженою ніж C++