Національний технічний університет України
«Київський політехнічний інститут»

Фізико технічний інститут

Кафедра математичних методів захисту інформації

МЕТОДИ РЕАЛІЗАЦІЇ КРИПТОГРАФІЧНИХ МЕХАНІЗМІВ

Лабораторна робота №1

Тема: “Вибір та реалізація базових фреймворків та бібліотек”

Виконали:
Драга Владислав ФІ-12мп
Чіхладзе Вахтанг ФІ-12мн

Перевірила
Селюх П.В.

Київ 2021

Мета роботи: 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.
Підгрупа 2B. Порівняння бібліотек OpenSSL, crypto++, CryptoLib, PyCrypto для розробки гібрідної криптосистеми під Linux платформу.

Посилання на репозиторій з виконаною лабораторною роботою: https://github.com/d-laslo/MPKM/tree/master/lab1

Хід роботи

Для профілювання пам'яті на c++ використано valgrind.

Для профілювання часу роботи на c++/python використано внутрішні таймер, оскільки знайдені утиліти для профілювання на c++ або не показували час роботи.

Профілювання часу на python відбувалося за тим же принципом, що і на c++, аби їхні результати були співставні.
Для профілювання пам'яті на python використано memory_profiler
Для профілювання достатньо виконати команду :
$ make
і результати профілювання будуть в директорії prof.
Для запуску програм має бути встановлено утиліти: make, g++

CryptoLib було завантажено з репозиторію https://github.com/cryptomator/cryptolib та запущено тестові скрипти в середовищі розробки IntellijIDEA.

Бібліотеку crypto++ було завантажено https://cryptopp.com/ .

Для тестування і профілювання бібліотеки OpenSSL має бути встановлено бібліотеки: (c++) libcrypto++-dev, libssl-dev. Дані бібліотеки було завантажено з репозиторію https://github.com/openssl/openssl.

Щоб профілювати бібліотеку PyCrypto, мають бути встановлені: python3, PyCryptodome(PyCrypto), memory-profiler . Дані бібліотеки були встановленні через утиліту pip.

Профілювання функцій бібліотеки OpenSSL:

Опис алгоритму генерації ключа:

1. Оголосити змінну, що містить довжину ключа.
2. Об’явити екземпляр структури RSA
3. Об’явити відкритий параметр e та надати його екземпляру структури BIGNUM.
4. викликати функцію `RSA_generate_key_ex` та передати екземпляр структури RSA, довжину ключа, екземпляр структури BIGNUM, та посилання на callback функцію(за замовчуванням NULL)

В результаті екземпляр структури RSA буде містити відкритий і закритий ключі.

Пам’ять:

```
==3004== Memcheck, a memory error detector
==3004== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==3004== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==3004== Command: ./obj/OpenSSL_size
==3004==
==3004==
==3004== HEAP SUMMARY:
==3004==     in use at exit: 4,496 bytes in 39 blocks
==3004==   total heap usage: 1,363,871 allocs, 1,363,832 frees, 255,989,350 bytes
allocated
==3004==
==3004== LEAK SUMMARY:
==3004==    definitely lost: 200 bytes in 2 blocks
==3004==    indirectly lost: 4,296 bytes in 37 blocks
==3004==      possibly lost: 0 bytes in 0 blocks
==3004==    still reachable: 0 bytes in 0 blocks
==3004==         suppressed: 0 bytes in 0 blocks
==3004== Rerun with --leak-check=full to see details of leaked memory
==3004==
==3004== For lists of detected and suppressed errors, rerun with: -s
==3004== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Час:

private/open key: 63.8842

ciphertext: 0.0190879
plaintext: 0.528481

Профілювання функцій бібліотеки PyCrypto

Опис алгоритму генерації ключа:

1. Задати довжину ключа як цілочисельну змінну
2. Задати параметр e як цілочисельну змінну
3. Викликати функцію `RSA.generate` з параметрами довжиною ключа та параметром e

На виході функція повертає об’єкт, що містить в собі пару відкритого і закритого ключів.

Пам’ять:
Filename: PyCrypto_size.py

```
Line #    Mem usage    Increment  Occurences   Line Contents
=============================================================
     6     40.8 MiB     40.8 MiB           1   @profile
     7                                         def memProf():
     8     40.8 MiB      0.0 MiB           1       msg = bytes(Settings.MESSAGE, 'utf-8')
     9    492.8 MiB      0.0 MiB        1001       for i in range(Settings.NUM_TESTS):
    10                                             # згенерувати відкритий/закритий ключ
    11    492.8 MiB    443.8 MiB        1000           key = RSA.generate(Settings.CRYPTO_RSA_KEY_LEN, e=17)
    12
    13                                             # зашифрувати повідомлення
    14    492.8 MiB      0.0 MiB        1000           encryptor = PKCS1_OAEP.new(key.publickey())
    15    492.8 MiB      3.1 MiB        1000           encrypted = encryptor.encrypt(msg)
    16
    17                                             # розшифрувати повідомлення
    18    492.8 MiB      0.0 MiB        1000           decryptor = PKCS1_OAEP.new(key)
    19    492.8 MiB      5.2 MiB        1000           decrypted = decryptor.decrypt(encrypted)
```

Час:
private/open key: 448.97037267684937
cipher text: 0.7617835998535156
open text: 2.2171568870544434

Профілювання функцій бібліотеки crypto++

Опис алгоритму генерації ключа:

1. Задати довжину ключа
2. Об’явити екземпляр класу рандомних чисел
3. Об’явити екземпляр класу приватного ключа
4. Об’явити екземпляр класу публічного ключа
5. У екземпляра класу приватного ключа викликати функцію `GenerateRandomWithKeySize` і передати параметри екземпляру класу рандомних чисел та довжини ключа
6. у екземпляра класу публічного ключа викликати функцію `Initialize` і передати модуль приватного ключа та експоненту приватного ключа

Таким чином буде ініціалізовано пару ключів, які зберігають своє значення у екземплярів класів.

Пам’ять:
==2943== Memcheck, a memory error detector
==2943== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==2943== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==2943== Command: ./obj/crypto++_size
==2943==
==2943==
==2943== HEAP SUMMARY:
==2943== in use at exit: 7,286 bytes in 10 blocks
==2943== total heap usage: 12,628,367 allocs, 12,628,357 frees, 1,467,277,560 bytes allocated
==2943==
==2943== LEAK SUMMARY:
==2943== definitely lost: 0 bytes in 0 blocks
==2943== indirectly lost: 0 bytes in 0 blocks
==2943== possibly lost: 0 bytes in 0 blocks
==2943== still reachable: 7,286 bytes in 10 blocks
==2943== suppressed: 0 bytes in 0 blocks
==2943== Rerun with --leak-check=full to see details of leaked memory
==2943==
==2943== For lists of detected and suppressed errors, rerun with: -s
==2943== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
Час:
pritate key: 30.2102
open key: 5.5043e-05
ciphertext: 0.00896258
plaintext: 0.833466

Висновок: Отже, ми ознайомились з фреймворками, що реалізовують криптографічні перетворення. Під час ознайомлення найлегшим було написати програму для профілювання, користуючись бібліотеками OpenSSL та PyCrypto. Але результати профілювання crypto++ виявили значно кращі результати ніж OpenSSL. Слід також зауважити, що crypto++ написано на C++, а OpenSSL на C. А PyCrypto виявився дуже повільним для криптографічних перетворень, оскільки python є високорівневою мовою програмування, відносно низькорівневих C/C++. Особиста оцінка по легкості написання програм можна показати у зростаючому порядку: PyCrypto, crypto++, OpenSSL, де найлегшим в

написанні був PyCrypto, а найскладнішим був OpenSSL. В результаті невеликого конкурсу, ми обрали переможцем бібліотеку PyCrypto, через його легкість використання.