Лабораторна робота №1

«Вибір та реалізація базових фреймворків та бібліотек»

Чашницька М., Ткаченко А., Ковалевський О.

5 грудня 2021 р.

Мета роботи: «Порівняння бібліотек PyCrypto, OpenSSL, crypto++ та CryptoLib за часом та використаної пам'ятті в залежності від довжини ключа/повідомлення для ОС Linux»

1 Хід роботи

1. Описати функціонал та сферу використання кожної із бібліотек;
2. Дослідити наявні функції симметричної/асимметричної криптографії - шифрування, підпис, геш-функції, генерування випадкових послідовностей;
3. Пояснити вибір бібліотеки для реалізації гібридної криптосистеми під платформу Linux.

2 Теоретична частина

2.1 OpenSSL

Дана бібліотека є open source реалізацією двох протоколів: Secure Sockets Layer (SSL) та Transport Layer Security (TLS). OpenSSL має інструменти для генерування приватних ключів RSA та Certificate Signing Requests (CSR-запитів), управлінням сертифікатів та кодуванням/декодуванням. Бібліотека написана на мові C із підтриманням оболонок під інші мови програмування.
Проект OpenSSL розробляє та підтримує програмне забезпечення OpenSSL – надійний повнофункціональний набір інструментів комерційного рівня для криптографії загального призначення та безпечного зв'язку. Прийняття технічних рішень у проекті керується Технічним комітетом OpenSSL (OTC), а управління проектом — Комітетом з управління OpenSSL (OMC). Проект діє відповідно до офіційного Статуту.

2.2 Cryptlib

Cryptlib — це кросплатформова криптографічна бібліотека із відкритим початковим кодом. Ця бібліотека забезпечує на найвищому рівні реалізацію повної безпеки таких послуг: S/MIME, PGP/openpgp, безпечних сесій протоколів SSL/TLS и SSH, сервісу Центрів сертифікації CMP, SCEP, RTCS, и OCSP.

Сфери використання

Бібліотека написана на низькорівневій мові програмування, отже її можна використовувати для реалізації криптографічних систем для мікроконтролерів та вбудованих систем. Немає чіткої спеціалізації, як наприклад Crypto++, проте відрізняється типом ліцензування.

Наявні алгоритми

- Алгоритми шифрування: AES, Blowfish, CAST-128, DES, TripleDES, IDEA, RC2, RC4, RC5, Skipjack.
- Алгоритми гешування: MD2, MD4, MD5, RIPEMD, SHA-1, SHA-2/256.

- Алгоритми MAC(імітовставки): HMAC-MD5, HMAC-RIPEMD-160, HMAC-SHA-1, HMAC-SHA-2.

- Відкриті ключі: Diffie-Hellman, DSA, ECDSA, ECDH, Elgamal, RSA.

2.3 Crypto++

Crypto++ (також відома як CryptoPP, libcrypto++ та libcryptopp) — це безкоштовна бібліотека C++ з відкритим вихідним кодом криптографічних алгоритмів та схем, написана китайським комп'ютерним інженером Вей Даєм.
Будучи випущеною в 1995, бібліотека повністю підтримує 32-розрядні та 64-розрядні архітектури для багатьох головних операційних систем та платформ, таких як Android (з використанням), Apple (Mac OS X та iOS), BSD, Cygwin, IBM AIX та S/ 390, Linux, MinGW, Windows, Windows Phone і Windows RT. Проект також підтримує компіляцію з використанням бібліотек різних середовищ виконання C++03, C++11 та C++17; і безліч інших компіляторів і IDE, що включають Borland Turbo C++, Borland C++ Builder, Clang, CodeWarrior Pro, GCC (з використанням GCC від Apple), Intel C++ Compiler (ICC), Microsoft Visual C/C++.

Наявні алгоритми

- схеми шифрування з автентифікацією: GCM, CCM, EAX, ChaCha20Poly1305, XChaCha20Poly1305

- високошвидкісні потокові шифри: ChaCha (8/12/20), ChaCha (IETF) HC (128/256), Panama, Rabbit (128/256), Sosemanuk, Salsa20 (8/12/20), XChaCha (8/12) /20), XSalsa20

- кандидати в AES і AES: AES (Rijndael), RC6, MARS, Twofish, Serpent, CAST-256

- інші блокові шифри: ARIA, Blowfish, Camellia, CHAM, HIGHT, IDEA, Kalyna (128/256/512), LEA, SEED, RC5, SHACAL-2, SIMECK, SIMON (64/128), Skipjack, SPECK (64/128), Simeck , SM4,Threefish (256/512/1024), Triple-DES (DES-EDE2 і DES-EDE3), TEA, XTEA

- Режими роботи блочних шифрів: ECB, CBC, CBC крадіжка шифрованого тексту (CTS), CFB, OFB, режим лічильника (CTR), XTS

- коди автентифікації повідомлень: BLAKE2b, BLAKE2s, CMAC, CBC-MAC, DMAC, GMAC (GCM), HMAC, Poly1305, SipHash, Two-Track-MAC, VMAC

- геш-функції: BLAKE2b, BLAKE2s, Keccack (F1600), SHA-1, SHA-2, SHA-3, SHAKE (128/256), SipHash, LSH (128/256), Tiger, RIPEMD (128/160/256/ 320), SM3, WHIRLPOOL

- криптографія з відкритим ключем: RSA, DSA, визначальний DSA (RFC 6979), ElGamal, Nyberg-Rueppel (NR), Rabin-Williams (RW), німецький цифровий підпис на основі EC (ECGDSA), LUC, LUCELG, DLIES (варіанти DHAES), ESIGN

- схеми заповнення для криптосистем з відкритим ключем: PKCS1 v2.0, OAEP, PSS, PSSR, IEEE P1363 EMSA2 і EMSA5

- криптографічні протоколи для узгодження ключів: Diffie-Hellman (DH), Unified Diffie-Hellman (DH2), Menezes-Qu-Vanstone (MQV), Hashed MQV (HMQV), Fully Hashed MQV (FHMQV), LUCDIF, XTR-DH

- криптографія на основі еліптичних кривих: ECDSA, Determinsitic ECDSA (RFC 6979), ed25519, ECGDSA, ECNR, ECIES, x25519, ECDH, ECMQV

- небезпечні або застарілі алгоритми, збережені для зворотної сумісності та історичної цінності: MD2, MD4, MD5, Panama Hash, DES, ARC4, SEAL 3.0, WAKE-OFB, DESX (DES-XEX3), RC2, SAFER, 3-WAY, GOST, SHARK , CAST-128, Squar

Функціонал даної бібліотеки опирається на принципи ООП. Ця архітектурна особливість дає змогу кінцевому користувачу спростити собі роботу, але залишає можливість модифікувати цільовий алгоритм.

Особливості роботи з бібліотекою

У загальному випадку дана бібліотека - це декілка принципів і класів від яких успадковані практично всі інші класи бібліотеки. Ієрархія класів та самі класи побудовані таким чином, щоб комбінування послідовності алгоритмів було якомога зручним для кінцевого користувача.

```
FileSource
  (inFileName,
   true,
   new HexDecoder
     (new StreamTransformationFilter
        (Decryptor,
         new HexEncoder (new FileSink (outFileName))
        )
     )
  );
```

Рис. 1: Приклад використання класів бібліотеки Crypto++

- беруться дані, що знаходяться у **inFileName**;
- за допомогою деякого поточного алгоритму перетворення, що задається об'єктом **Decryptor**, дані переводяться із шестнадцяткового виду у двійковий;
- перетворюється назад у шістнадцяткове представлення;
- зберігається у файлі **OutFileName**.

На перший погляд може здатися, що тут виконується багато дій, але замість конструювання декількох статичних об'єктів, читання із файлу, контролю розшифрування, переведення із різних систем числення, було реалізовано лише один статичний об'єкт та декілька динамічних, що й зробили всю роботу, причому проконтролювавши всі передані їм об'єкти/параметри на правильність.

У розрізі бібліотеки це називається *Pipelining*. Це парадигма, що дозволяє направляти потік даних через набір фільтрів, які певним чином змінюють дані, від джерела даних до приймача.

Рис. 2: Графічна аналогія парадигми *Pipelining*

```
FileSource (filename,
            new Base64Encoder (new FileSink (filename + ".b64")));
```

Рис. 3: Набір фільтрів у Crypto++

Ця парадигма може також використовуватися у зв'язці з механізмом *ChannelSwitch*. Він побудован на классі, що добавляю підтримку концепції *Read-Once Write-Many*:

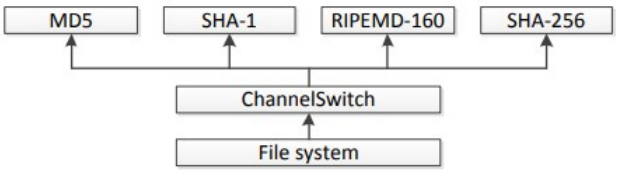

Рис. 4: Механізм *ChannelSwitch*

Приклади використання статичних та динамічних об'єктів (Рис.5 та Рис.6)

```
std::string digest, hex_digest;
...
HexEncoder encoder;
encoder.Attach (new StringSink (hex_digest));
encoder.Put ((byte *) digest.c_str (), digest.length ());
encoder.MessageEnd ();
```

Рис. 5: Статичні об'єкти

```
std::string digest, hex_digest;
...
StringSource (digest,
              true,
              new HexEncoder (new StringSink (hex_digest))
             );
```

Рис. 6: Динамічні об'єкти

```
CFB_Mode<DES_EDE3>::Encryption Encryptor;
CFB_Mode<AES>::Encryption Encryptor;
CFB_Mode<Blowfish>::Encryption Encryptor;
```

Рис. 7: Приклади шаблонів

Також, важливою особливістю бібліотеки є використання шаблонів. Цей механізм C++ використовується для всіх класів алгоритмів, параметри яких можна варіювати. Для прикладу можна розглянути декілька алгоритмів блочного шифрування (Рис. 7)

Підсумовуючи, можна виділити дві основні властивості бібліотеки:

1. Практично всі класи пов'язані між собою або можна реалізувати їх роботу в конвеєрному режимі.

2. Використовуючи шаблони, можна змінювати параметри, режими та деякі інші аспекти роботи алгоритмів.

Особливий функціонал бібліотеки

Фільтри трансформації. Класи, що використовуються в якості побудови фільтрів для об'єктів, основний функціонал яких не пов'язаний із конвеєрним використанням:

- **HashFilter** — фільтри на основі геш-алгоритму;
- **PK EncryptorFilter/PK DecryptorFilter** — фільтри перетворень для алгоритмів із публічним ключем;
- **SignerFilter** — фільтри для отримання підпису даних у алгортмах ЦП;
- **SignatureVerificationFilter** — фільтри для верифікації підпису даних у алгоритмах ЦП;
- **StreamTransformationFilter** — фільтри для роботи із семитричними алгоритмами.

2.4 PyCrypto

PyCryptodome — це автономний Python пакет низькорівневих криптографічних примітивів. Він підтримує Python 2.7, Python 3.5 і більш нові версії, а також PyPy. Для швидшої роботи алгоритмів з відкритим ключем у системах Unix вам слід встановити GMP у вашій системі. PyCryptodome — це форк PyCrypto. PyCryptodome не є обгорткою окремої бібліотеки C, як OpenSSL. Більшість алгоритмів реалізовано на чистому Python. Тільки ті частини, для яких надзвичайно важливою є їхня швидкодія (наприклад, блочні шифри), реалізуються як розширення C.

3 Практична частина

Тестування обраних бібліотек відбувалися за такими критеріями: час виконання операції та об'єм використаної пам'ятті в залежності від довжини ключа/повідомлмення.

3.1 OpenSSL

Перший етап аналізу - це застосування операції «speed» для заміру часу роботи найбільш популярних криптопримітивів (AES, RSA, SHA-256, SHA-512):

```
Doing sha256 for 3s on 16 size blocks: 12977216 sha256's in 2.99s
Doing sha256 for 3s on 64 size blocks: 7374919 sha256's in 2.99s
Doing sha256 for 3s on 256 size blocks: 3429656 sha256's in 2.99s
Doing sha256 for 3s on 1024 size blocks: 1083009 sha256's in 2.99s
Doing sha256 for 3s on 8192 size blocks: 145772 sha256's in 3.00s
Doing sha256 for 3s on 16384 size blocks: 71871 sha256's in 2.99s
Doing sha512 for 3s on 16 size blocks: 9023077 sha512's in 2.99s
Doing sha512 for 3s on 64 size blocks: 9081303 sha512's in 2.99s
Doing sha512 for 3s on 256 size blocks: 3913032 sha512's in 2.99s
Doing sha512 for 3s on 1024 size blocks: 1435349 sha512's in 2.99s
Doing sha512 for 3s on 8192 size blocks: 213890 sha512's in 2.99s
Doing sha512 for 3s on 16384 size blocks: 108514 sha512's in 3.00s
Doing aes-128 cbc for 3s on 16 size blocks: 22416788 aes-128 cbc's in 2.99s
Doing aes-128 cbc for 3s on 64 size blocks: 6345486 aes-128 cbc's in 2.99s
Doing aes-128 cbc for 3s on 256 size blocks: 1612666 aes-128 cbc's in 2.99s
Doing aes-128 cbc for 3s on 1024 size blocks: 404677 aes-128 cbc's in 3.00s
Doing aes-128 cbc for 3s on 8192 size blocks: 51590 aes-128 cbc's in 3.00s
Doing aes-128 cbc for 3s on 16384 size blocks: 25786 aes-128 cbc's in 3.00s
Doing aes-192 cbc for 3s on 16 size blocks: 19265203 aes-192 cbc's in 2.99s
Doing aes-192 cbc for 3s on 64 size blocks: 5198287 aes-192 cbc's in 3.00s
Doing aes-192 cbc for 3s on 256 size blocks: 1345667 aes-192 cbc's in 3.00s
Doing aes-192 cbc for 3s on 1024 size blocks: 341531 aes-192 cbc's in 3.00s
Doing aes-192 cbc for 3s on 8192 size blocks: 42543 aes-192 cbc's in 3.00s
Doing aes-192 cbc for 3s on 16384 size blocks: 21525 aes-192 cbc's in 3.00s
Doing aes-256 cbc for 3s on 16 size blocks: 16734616 aes-256 cbc's in 2.99s
Doing aes-256 cbc for 3s on 64 size blocks: 4486475 aes-256 cbc's in 3.00s
Doing aes-256 cbc for 3s on 256 size blocks: 1149938 aes-256 cbc's in 3.00s
Doing aes-256 cbc for 3s on 1024 size blocks: 289225 aes-256 cbc's in 3.00s
Doing aes-256 cbc for 3s on 8192 size blocks: 36273 aes-256 cbc's in 3.00s
Doing aes-256 cbc for 3s on 16384 size blocks: 18193 aes-256 cbc's in 2.99s
Doing 512 bits private rsa's for 10s: 206068 512 bits private RSA's in 10.00s
Doing 512 bits public rsa's for 10s: 3744876 512 bits public RSA's in 10.00s
Doing 1024 bits private rsa's for 10s: 95834 1024 bits private RSA's in 9.98s
Doing 1024 bits public rsa's for 10s: 1575945 1024 bits public RSA's in 10.00s
Doing 2048 bits private rsa's for 10s: 14984 2048 bits private RSA's in 10.00s
Doing 2048 bits public rsa's for 10s: 514129 2048 bits public RSA's in 9.99s
Doing 3072 bits private rsa's for 10s: 4966 3072 bits private RSA's in 9.99s
Doing 3072 bits public rsa's for 10s: 246721 3072 bits public RSA's in 10.00s
Doing 4096 bits private rsa's for 10s: 2237 4096 bits private RSA's in 10.00s
Doing 4096 bits public rsa's for 10s: 145398 4096 bits public RSA's in 10.00s
Doing 7680 bits private rsa's for 10s: 243 7680 bits private RSA's in 10.02s
Doing 7680 bits public rsa's for 10s: 43198 7680 bits public RSA's in 10.00s
Doing 15360 bits private rsa's for 10s: 48 15360 bits private RSA's in 10.10s
Doing 15360 bits public rsa's for 10s: 11116 15360 bits public RSA's in 9.99s
OpenSSL 1.1.1  11 Sep 2018
built on: Mon Aug 23 17:02:39 2021 UTC
options:bn(64,64) rc4(16x,int) des(int) aes(partial) blowfish(ptr)
compiler: gcc -fPIC -pthread -m64 -Wa,--noexecstack -Wall -Wa,--noexecstack -g -O2 -fdebug-pr
The 'numbers' are in 1000s of bytes per second processed.
type             16 bytes     64 bytes    256 bytes   1024 bytes   8192 bytes  16384 bytes
aes-128 cbc     119956.06k   135823.11k   138074.41k   138129.75k   140875.09k   140825.94k
aes-192 cbc     103091.39k   110896.79k   114830.25k   116575.91k   116170.75k   117555.20k
aes-256 cbc      89549.78k    95711.47k    98128.04k    98722.13k    99049.47k    99690.34k
sha256           69443.30k   157857.80k   293642.79k   370903.42k   398054.74k   393824.24k
sha512           48284.02k   194382.41k   335028.83k   491571.03k   586015.68k   592631.13k
                  sign    verify    sign/s verify/s
rsa  512 bits 0.000049s 0.000003s  20606.8 374487.6
```

Рис. 8: Швидкість роботи AES, RSA, SHA-256, SHA-512

Для аналізу використаної пам'ятті ми використали *python*-бібліотеку *pyOpenSSL* за допомогою якої було реалізовано (як приклад) генерацію пари ключів RSA та бібліотеку *memory profiler*, що дозволяє аналізувати кількість виділеної памя'тті на кожну інструкцію:

```
Line #    Mem usage    Increment  Occurences   Line Contents
=============================================================
    11    115.8 MiB    115.8 MiB           1   @profile
    12                                         def rsa_key_gen_pair():
    13    115.8 MiB      0.0 MiB           1       k = crypto.PKey()
    14    115.9 MiB      0.2 MiB           1       key = k.generate_key(crypto.TYPE_RSA, 2048)
    15
    16    115.9 MiB      0.0 MiB           1       return key
```

Рис. 9: Використана пам'ять при генерації пари ключів RSA

3.2 Crypto++

Швидкість роботи та використана пам'ять базових алгоритмів:

1. 8 byte:
   - RSA: 944KB, 0.00980505 s
   - AES-256: 956KB, 0.000103321 s
   - SHA-256: 460KB, 0.00000338618 s
2. 16 byte:
   - RSA: 952KB, 0.0106922 s
   - AES-256: 1.5MB, 0.00011149 s
   - SHA-256: 464KB, 0.00000370498 s
3. 32 byte:
   - RSA: 968KB, 0.0115434 s
   - AES-256: 1.6MB, 0.00016686 s
   - SHA-256: 476KB, 0.00000494023 s

3.3 PyCrypto

Для тестування AES, SHA-256, SHA-512 використовувалися блоки довжиною $16, 64, 256, 1024, 8192, 16384$. Для RSA — блоки довжиною $1024, 2048, 4096, 8192$. Отримані результати:

```
The slowest run took 273.26 times longer than the fastest. This could mean that an intermediate result is being cached.
100000 loops, best of 5: 4.45 µs per loop
The slowest run took 4.86 times longer than the fastest. This could mean that an intermediate result is being cached.
100000 loops, best of 5: 4.55 µs per loop
The slowest run took 4.53 times longer than the fastest. This could mean that an intermediate result is being cached.
100000 loops, best of 5: 5.12 µs per loop
The slowest run took 5.63 times longer than the fastest. This could mean that an intermediate result is being cached.
100000 loops, best of 5: 6.76 µs per loop
The slowest run took 4.86 times longer than the fastest. This could mean that an intermediate result is being cached.
10000 loops, best of 5: 22.2 µs per loop
10000 loops, best of 5: 40.5 µs per loop

[ ] measures_AES

[0.4753024566009117,
 0.47111643300013384,
 0.5202135963998444,
 0.6977971226006048,
 0.22632523420033976,
 0.40765817260107723]
```

Рис. 10: Швидкість шифрування AES

```
The slowest run took 68.21 times longer than the fastest. This could mean that an intermediate result is being cached.
100000 loops, best of 5: 2.49 µs per loop
The slowest run took 5.32 times longer than the fastest. This could mean that an intermediate result is being cached.
100000 loops, best of 5: 2.45 µs per loop
The slowest run took 6.44 times longer than the fastest. This could mean that an intermediate result is being cached.
100000 loops, best of 5: 2.53 µs per loop
The slowest run took 6.14 times longer than the fastest. This could mean that an intermediate result is being cached.
100000 loops, best of 5: 2.49 µs per loop
The slowest run took 6.33 times longer than the fastest. This could mean that an intermediate result is being cached.
100000 loops, best of 5: 2.48 µs per loop
The slowest run took 6.13 times longer than the fastest. This could mean that an intermediate result is being cached.
100000 loops, best of 5: 2.47 µs per loop
```

```
[ ] measures_SHA256
```

```
[0.2557844129994919,
 0.2516272819993901,
 0.25525778719966186,
 0.2542920400002913,
 0.2570922049984802,
 0.2516652723999869]
```

Рис. 11: Швидкість SHA-256

```
The slowest run took 39.21 times longer than the fastest. This could mean that an intermediate result is being cached.
100000 loops, best of 5: 2.61 µs per loop
The slowest run took 6.07 times longer than the fastest. This could mean that an intermediate result is being cached.
100000 loops, best of 5: 2.58 µs per loop
The slowest run took 5.36 times longer than the fastest. This could mean that an intermediate result is being cached.
100000 loops, best of 5: 2.59 µs per loop
The slowest run took 5.95 times longer than the fastest. This could mean that an intermediate result is being cached.
100000 loops, best of 5: 2.6 µs per loop
The slowest run took 5.69 times longer than the fastest. This could mean that an intermediate result is being cached.
100000 loops, best of 5: 2.64 µs per loop
The slowest run took 5.54 times longer than the fastest. This could mean that an intermediate result is being cached.
100000 loops, best of 5: 2.6 µs per loop
```

```
[ ] measures_SHA512
```

```
[0.26738396900036604,
 0.2647209078000742,
 0.2690132652001921,
 0.2643862063996494,
 0.2674928758009628,
 0.267132725600095]
```

Рис. 12: Швидкість SHA-512

```
10 loops, best of 5: 141 ms per loop
1 loop, best of 5: 380 ms per loop
The slowest run took 4.87 times longer than the fastest. This could mean that an intermediate result is being cached.
1 loop, best of 5: 1.9 s per loop
The slowest run took 8.13 times longer than the fastest. This could mean that an intermediate result is being cached.
1 loop, best of 5: 13.7 s per loop
```

```
[ ] keypair_measures_RSA
```

```
[1.5264911545993527, 0.7257511872005125, 3.6195750465994934, 51.91353893199994]
```

Рис. 13: Швидкість створення пари ключів RSA

```
1000 loops, best of 5: 1.15 ms per loop
1000 loops, best of 5: 445 µs per loop
100 loops, best of 5: 2.72 ms per loop
1000 loops, best of 5: 860 µs per loop
100 loops, best of 5: 10.2 ms per loop
100 loops, best of 5: 1.89 ms per loop
10 loops, best of 5: 52.9 ms per loop
100 loops, best of 5: 4.89 ms per loop
```

```
[ ] signature_measures
```

```
[1.1785568484003306,
 0.2772046040001442,
 1.0314513619996433,
 0.5386263647997112]
```

```
[ ] verify_measures
```

```
[0.4658859688002849, 0.8673194899994996, 0.193736295200506, 0.4934858251988771]
```

Рис. 14: Швидкість підпису та перевірки підпису RSA

Висновки

Щодо швидкості криптографічних перетворень PyCrypto поступається Crypto++ за концепцією мови - високорівневості. Але з точки зору порогу входження по складності, ми би обрали для реалізації гібридної криптосистеми бібліотку PyCrypto.

В процесі виконання лабораторної роботи виникли складнощі по пошуку інформації про практичне застосування бібліотеки CryptoLib: в основному це були певні вузьконаправлені пакети із реалізаціями певних криптографічних алгоритмів. Тому, враховуючи, із якими кандидатами було порівняння, ми відкинули дану бібліотеку.