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:
Корж Нікіта ФІ-12мн
Тафтай Анастасія ФІ-12мп
Мазур Анастасія 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.
Порівняння бібліотек OpenSSL, crypto++, CryptoLib, PyCrypto для розробки гібрідної криптосистеми під Windows платформу.

Хід роботи:

Pycryptodome

PyCryptodome - це автономний пакет низькорівневих криптографічних примітивів Python, а також є розширенням PyCrypto. Він вносить такі вдосконалення щодо останньої офіційної версії PyCrypto (2.6.1):

1. Аутентифіковані режими шифрування (GCM, CCM, EAX, SIV, OCB)
2. Прискорений AES на платформах Intel через AES-NI
3. Першокласна підтримка PyPy
4. Криптографія з еліптичними кривими (NIST P-256, P-384 та P-521)
5. Кращий та компактніший API
6. SHA-3 (включаючи SHAKE та cSHAKE XOF), усічені алгоритми хешування SHA-512 та BLAKE2
7. Потокові шифри Salsa20 та ChaCha20/XChaCha20
8. Poly1305 MAC
9. Аутентифіковані шифри ChaCha20-Poly1305 та XChaCha20-Poly1305
10. функції виведення scrypt, bcrypt та HKDF
11. Детермінований (EC) DSA
12. Захищені паролем контейнери ключів PKCS#8
13. Схема секретного обміну Шаміра
14. Випадкові числа отримуються безпосередньо з ОС (а не з CSPRNG у просторі користувача)
15. Спрощений процес встановлення, включаючи кращу підтримку Windows

PyCryptodome не є обгорткою для окремої бібліотеки C, такої як OpenSSL. Наскільки це можливо, алгоритми реалізовані на чистому Python. Тільки фрагменти, які є надзвичайно важливими для продуктивності (наприклад, блок -шифри), реалізуються як розширення C.

Для виконання цієї частини роботи було імпортовано бібліотеку PyCryptodome, memory_profiler та time. Memory_profiler був використаний для того, щоб порахувати кількість використаної пам’яті для окремих функцій, а time був використаний для того, щоб порахувати кількість витраченого часу на виконання окремих функцій.

Симетрична криптографія

1) AES (EAX)

Для тестування AES з сучасним модом EAX був використаний текст довжини 256
Час для шифрування тексту з memory_profiler : 0.06017446517944336
Час для шифрування тексту без memory_profiler : 0.001994609832763672

```
Line #    Mem usage    Increment  Occurences   Line Contents
=============================================================
    29     23.0 MiB     23.0 MiB           1   @memory_profiler.profile
    30                                         def encryptEAX(text):
    31     23.1 MiB      0.0 MiB           1       cipher = AES.new(key, AES.MODE_EAX)
    32     23.1 MiB      0.0 MiB           1       nonce = cipher.nonce
    33     23.1 MiB      0.0 MiB           1       ciphertext, tag = cipher.encrypt_and_digest(text.encode('ascii'))
    34     23.1 MiB      0.0 MiB           1       return nonce, ciphertext, tag


0.06017446517944336 seconds for encryption
```

Час для дешифрування тексту з memory_profiler: 0.01000213623046875
Час для дешифрування тексту без memory_profiler : 0.0019941329956054688

```
Line #    Mem usage    Increment  Occurences   Line Contents
=============================================================
    37     23.1 MiB     23.1 MiB           1   @memory_profiler.profile
    38                                         def decryptEAX(nonce, ciphertext, tag):
    39     23.1 MiB      0.0 MiB           1       cipher = AES.new(key, AES.MODE_EAX, nonce=nonce)
    40     23.1 MiB      0.0 MiB           1       plaintext = cipher.decrypt(ciphertext)
    41     23.1 MiB      0.0 MiB           1       try:
    42     23.1 MiB      0.0 MiB           1           cipher.verify(tag)
    43     23.1 MiB      0.0 MiB           1           return plaintext.decode('ascii'), time.time() - start_time
    44                                             except:
    45                                                 return False


0.01000213623046875 seconds for decryption
```

2) AES(CFB)

Для тестування AES з модом CFB був використаний текст довжини 256.
Час для шифрування тексту з memory_profiler : 0.09994697570800781
Час для шифрування тексту без memory_profiler : 0.001994609832763672

```
Line #    Mem usage    Increment  Occurences   Line Contents
=============================================================
    65     22.8 MiB     22.8 MiB           1   @memory_profiler.profile
    66                                         def encryptCFB(plaintext):
    67     22.8 MiB      0.0 MiB           1       data_bytes = bytes(plaintext, 'utf-8')
    68     22.8 MiB      0.0 MiB           1       padded_bytes = pad(data_bytes, AES.block_size)
    69     22.8 MiB      0.0 MiB           1       AES_obj = AES.new(key, AES.MODE_CFB, iv)
    70     22.8 MiB      0.0 MiB           1       ciphertext = AES_obj.encrypt(padded_bytes)
    71     22.8 MiB      0.0 MiB           1       return ciphertext


0.09994697570800781 seconds for encryption
```

Час для дешифрування тексту з memory_profiler : 0.009994029998779297
Час для дешифрування тексту без memory_profiler : 0.0

```
Line #    Mem usage    Increment  Occurences   Line Contents
=============================================================
    74     22.8 MiB     22.8 MiB           1   @memory_profiler.profile
    75                                         def decryptCFB(ciphertext):
    76     22.8 MiB      0.0 MiB           1       AES_obj = AES.new(key, AES.MODE_CFB, iv)
    77     22.8 MiB      0.0 MiB           1       raw_bytes = AES_obj.decrypt(ciphertext)
    78     22.8 MiB      0.0 MiB           1       extracted_bytes = unpad(raw_bytes, AES.block_size)
    79     22.8 MiB      0.0 MiB           1       return extracted_bytes


0.009994029998779297 seconds for decryption
```

3) Salsa20 — це потоковий шифр, розроблений Деніелем Дж. Бернштейном. За бажанням секретний ключ має довжину 256 біт, але він також може працювати з 128-бітними ключами.

Для тестування Salsa20 був використаний текст довжини 256.
Час для шифрування тексту з memory_profiler : 0.08995413780212402
Час для шифрування тексту без memory_profiler : 0.0

```
Line #    Mem usage    Increment  Occurences   Line Contents
=============================================================
    98     22.8 MiB     22.8 MiB           1   @memory_profiler.profile
    99                                         def encryptSAlSA(plaintext):
   100     22.8 MiB      0.0 MiB           1       cipher = Salsa20.new(key=secret)
   101     22.8 MiB      0.0 MiB           1       msg = cipher.nonce + cipher.encrypt(plaintext)
   102     22.8 MiB      0.0 MiB           1       return msg


0.08995413780212402 seconds for encryption
```

Час для дешифрування тексту з memory_profiler : 0.01000714302062988
Час для дешифрування тексту без memory_profiler : 0.0

```
Line #    Mem usage    Increment  Occurences   Line Contents
=============================================================
   105     22.8 MiB     22.8 MiB           1   @memory_profiler.profile
   106                                         def decryptSAlSA(msg):
   107     22.8 MiB      0.0 MiB           1       msg_nonce = msg[:8]
   108     22.8 MiB      0.0 MiB           1       ciphertext = msg[8:]
   109     22.8 MiB      0.0 MiB           1       cipher = Salsa20.new(key=secret, nonce=msg_nonce)
   110     22.8 MiB      0.0 MiB           1       plaintext = cipher.decrypt(ciphertext)
   111     22.8 MiB      0.0 MiB           1       return plaintext


0.010007143020629883 seconds for decryption
```

Геш-функції:

Для гешування був обраний SHA-256. SHA-256 належить до сімейства криптографічних хешів SHA-2. Він створює 256-бітний дайджест повідомлення.
Був використаний hexdigest(). Повертає хеш-дайджест, розрахований на основі даних, оброблених досі. Шістнадцяткове кодування.

```
--------- Crypto.HASH256.Package ---------
185f8db32271fe25f561a6fc938b2e264306ec304eda518007d1764826381969
```

Асиметрична криптографія:

RSA є найбільш поширеним і використовуваним алгоритмом відкритого ключа. Його безпека заснована на складності розкладання великих цілих чисел на множники. Алгоритм витримує атаки більше 30 років, і тому вважається достатньо безпечним для нових конструкцій. Алгоритм можна використовувати як для конфіденційності (шифрування), так і для аутентифікації (цифровий підпис). Варто зазначити, що підписання та дешифрування значно повільніше, ніж перевірка та шифрування. Криптографічна міцність в першу чергу пов'язана з довжиною модуля RSA n. У 2017 році достатньою довжиною вважається 2048 біт. І шифротексти RSA, і підписи RSA такі ж великі, як модуль RSA n (256 байт, якщо n має довжину 2048 біт).

За допомогою модулю RSA було сгенеровано публічний та секретний ключ.
Генерація публічного та секретного ключа зайняла: 3.5275659561157227 seconds

```
Line #    Mem usage    Increment  Occurences   Line Contents
=============================================================
   137     22.9 MiB     22.9 MiB           1   @memory_profiler.profile
   138                                         def generate_keys():
   139     22.9 MiB      0.0 MiB           1       moduls_lenght = 256 * 4
   140     22.9 MiB      0.1 MiB           1       private_key = RSA.generate(moduls_lenght, Random.new().read)
   141     22.9 MiB      0.0 MiB           1       public_key = private_key.public_key()
   142     22.9 MiB      0.0 MiB           1       return private_key, public_key


3.5275659561157227 seconds
```

Г

Цифрові підписи:

Для цифрового підпису був використаний PKCS#1 v1.5 PSS (RSA) - стара, але все ще міцна схема цифрового підпису на основі RSA. В даному пункті було згенеровано приватний та публічний ключі. Було сформовано цифровий підпис, а також була проведена верифікація цифрового підпису.

```
--------- Crypto.Signature.Package ---------
7dff319aa9ddb03065aa7f2e2e8ffe3988645cf17d76c86d733d3c41b853e534b569635876d987ef8f63c39cb542b33e8271709 6d0369b2ee5105cb72a268efbce5fad3e5a163358
The signature is valid.
```

Як результат було повернено цифровий підпис та видно, що даний цифровий підпис пройшов верифікацію.

Bouncy Castle

Bouncy Castle — це програмна бібліотека, в якій представлений великий функціонал з області криптографії. Існують реалізації бібліотеки мови програмування Java та C#. Бібліотека включає в себе як реалізації великого числа криптографічних функцій, так і підтримку стандартних високорівневих криптографічних API відповідних платформ, а також містить низькорівневі пропрієтарні API для більш гнучкого та ефективного доступу до функціоналу.
В основі архітектури лежить набір низькорівневих API, які реалізують всі криптографічні алгоритми. Причина, з якої використовується саме низькорівневий API, полягає в тому, що в деяких пристроях, що працюють на платформі JavaME, дуже обмежені ресурси пам'яті, або коли доступ до бібліотеки JCE неможливий (така ситуація може виникнути, наприклад, при використанні аплетів).
Криптопровайдер, сумісний з JCE, побудований на низькорівневому API. Таким чином, вихідний код криптопровайдера JCE може служити прикладом того, як вирішити багато «нагальних» проблеми криптографії, використовуючи низькорівневий API.

Симетрична криптографія

Bouncy Castle підтримує широкий спектор алгоритмів симетричного шифрування з різними режимами та паддінгами.
Цими алгоритамми є;

Name	KeySizes (in bits)	Block Size
AES	0 .. 256 (192)	128 bit
AESWrap	0 .. 256 (192)	128 bit
ARIA	0 .. 256 (192)	128 bit
ARIAWrap	0 .. 256 (192)	128 bit
Blowfish	0 .. 448 (448)	64 bit
Camellia	128, 192, 256	128 bit
CamelliaWrap	128, 192, 256	128 bit
CAST5	0 .. 128(128)	64 bit
CAST6	0 .. 256(256)	128 bit
DES	64	64 bit
DESede	128, 192	64 bit
DESedeWrap	128, 192	128 bit
DSTU7624	128, 256, 512	128/256/512 bit
DSTU7624Wrap	128, 256, 512	128/256/512 bit
GCM	128, 192, 256(192)	AEAD Mode Cipher
GOST28147	256	64 bit
IDEA	128 (128)	64 bit

Noekeon	128(128)	128 bit
RC2	0 .. 1024 (128)	64 bit
RC5	0 .. 128 (128)	64 bit
RC5-64	0 .. 256 (256)	128 bit
RC6	0 .. 256 (128)	128 bit
Rijndael	0 .. 256 (192)	128 bit
SEED	128(128)	128 bit
SEEDWrap	128(128)	128 bit
Serpent	128, 192, 256 (256)	128 bit
Skipjack	0 .. 128 (128)	64 bit
SM4	128(128)	128 bit
TEA	128 (128)	64 bit
Threefish-256	256	256 bit
Threefish-512	512	512 bit
Threefish-1024	1024	1024 bit
Twofish	128, 192, 256 (256)	128 bit
XTEA	128 (128)	64 bit

Підтримувальні режими: ECB CBC OFB(n) CFB(n) SIC (also known as CTR) OpenPGPCFB CTS (equivalent to CBC/WithCTS) FF1 FF3-1 GOFB GCFB CCM (AEAD) EAX (AEAD) GCM (AEAD) GCM-SIV (AEAD) OCB (AEAD)
Паддінги: No padding PKCS5/7 ISO10126/ISO10126-2 ISO7816-4/ISO9797-1 X9.23/X923 TBC ZeroByte withCTS (if used with ECB mode)

Також є такі стрімові алгоритими шифрування:

Name	KeySizes (in bits)
RC4	40 .. 2048 bits (128)
HC128	-128
HC256	-256
ChaCha	128/256
Salsa20	128/256
XSalsa20	256
VMPC	128/6144(128)
Grainv1	80
Grain128	128
Zuc128	128
Zuc256	256

Слід зауважити, що в наявні укрїнські стандарти шифрування (шифр Калина (ДСТУ7624), геш функція Купина (ДСТУ7564), цифровий підпис ДСТУ 4145-2002)

Цифрові підписи

Прикладами цифрових підписів є XMSSMT-SHAKE128, SHA256withRSAandMGF1 RIPEMD160withRSA та багато інших. Повний перелік наявний за посиланням https://www.bouncycastle.org/specifications.html

Геш-функції

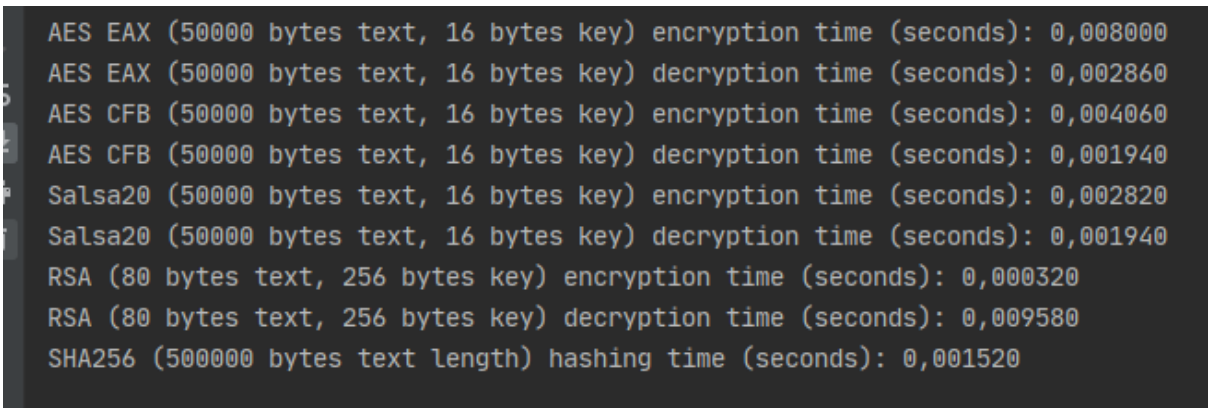
В бібліотеці є багато реалізацій геш функцій з різними параетрами, такі як наприклад SHA, MD, Keccak, Blake2b, DSTU7564, GOST3411,Haraka, RipeMD, Skein, Tiger, Whirlpool, SM3
А також такі алгоритми гешування паролів: BCrypt OpenBSDBcyrpt SCrypt PKCS5S1, any Digest, any symmetric Cipher, ASCII PKCS5S2, any HMac, any symmetric Cipher, ASCII, UTF8 PKCS12, any Digest, any symmetric Cipher, Unicode

Асиметрична криптографія

Підтримує RSA та ElGamal.

Результати виконання

На скриншоті можна побачити час виконання деяких алгоритмів шифрування. Шифрування запускалось 50 разв на випадково згенерованих текстах та ключах на компьютері з характреристиками процесора: Intel(R) Core(TM) i7-6500U CPU @ 2.50GHz 2.59 GHz



Crypto++ (або CryptoPP)

Crypto++ Library — це безкоштовна бібліотека класів криптографічних схем для C++. Бібліотека повністю підтримує 32-розрядні та 64-розрядні архітектури для багатьох основних операційних систем та платформ, таких як Android (із використанням STLport), Apple (Mac OS X та iOS), BSD, Cygwin, IBM AIX та S/390, Linux, MinGW, Solaris, Windows, Windows Phone та Windows RT. Проект також підтримує компіляцію за допомогою бібліотеки різних середніх виконання C ++ 03, C ++ 11 та C ++ 17; та багато інших компіляторів та IDE, що включають у себе Borland Turbo C ++, Borland C ++ Builder, Clang, CodeWarrior Pro, GCC (з використанням GCC від Apple), компілятор Intel C ++ (ICC), Microsoft Visual C/C ++.

Симетрична криптографія

AES (EAX)

Довжина ключа 128 біт, та текст довжини 50000 байт:

```
EAX AES encryption time: 0.001 seconds
EAX AES decryption time: 0 seconds
```

Довжина ключа 128 біт, та текст довжини 100кб:

```
EAX AES encryption time: 0.002 seconds
EAX AES decryption time: 0.002 seconds
```

AES (CFB)

Довжина ключа 128 біт, та текст довжини 50000 байт:

```
CFB AES encryption time: 0.001 seconds
CFB AES decryption time: 0.001 seconds
```

Довжина ключа 128 біт, та текст довжини 100кб:

```
CFB AES encryption time: 0.001 seconds
CFB AES decryption time: 0.001 seconds
```

Salsa20

Довжина ключа 128 біт, та текст довжини 50000 байт:

```
Salsa20 encryption time: 0 seconds
Salsa20 decryption time: 0 seconds
```

Довжина ключа 128 біт, та текст довжини 500кб:

```
Salsa20 encryption time: 0.002 seconds
Salsa20 decryption time: 0.001 seconds
```

Цифрові підписи

Crypto ++ підтримує дві об'ємні категорії цифрових підписів: схеми підписів з додатком (SSA) та схеми підписів з відновленням (PSSR). Схема підпису з додатком вимагає, щоб верифікатор мав три параметри: відкритий ключ, повідомлення та підпис. Схема підпису з відновленням вимагає лише відкритий ключа та підпис - повідомлення відновлюється з підпису.

Crypto++ підтримує схеми цифрового підпису з додатком: RSA, DSA, GDSA, ESIGN і Rabin-Williams.
А також такі схеми підпису з відновленням: RSA, Nyberg-Rueppel і Rabin-Williams.

Геш-функції

У бібліотеці класс SHA256 має розмір блоку 64 байти та повертає хеш-значення довжиною 32 байти.

Текст довжини 50000 байт:

```
SHA hasing time: 0 seconds
Digest: 41813D0F612BB7C875951B5BD3B36D750AD45DBCBC0CB51A27F729DD26CAE5FC
```

Текст довжини 1000 кб:

```
SHA hasing time: 0.005 seconds
Digest: BE85334377BF93EDCA6043D06799B4A73D26F1AF7E0936CF8D2026B55A6BA6E1
```

Асиметрична криптографія

Crypto ++ підтримує такі асиметричні алгоритми: RSA, ElGamal, Nyberg-Rueppel (NR), Rabin-Williams (RW), EC-based German Digital Signature (ECGDSA), та інші.

Довжина відкритого ключа 2048 біт, текст довжною 80 байт.
Для генерації секретних ключів був використаний метод GenerateRandomWithKeySize() з классу RSA::PrivateKey

```
RSA generation time: 0.053 seconds
RSA signing time: 0.003 seconds
Verification  time: 0.001 seconds
```

Висновок:

Під часа виконання роботи був проведенний аналіл 3 бібліток : Pycrypto, Bouncy Castle та Crypto++. Кожен з учасників групи представив результати роботи обраної бібліотеки вище. Був виміряний час роботи для багатьох функцій, а в Pycryptodome ще й кількість затраченої пам'яті. Було отримано, наступні результати (для розміру тексту 50 000 байт, та ключів однакової довжини):

Бібліотека	**Bouncy Castle**	**Crypto++**
AES (EAX)	encryption: 0,00586 seconds decryption: 0,00352 seconds	encryption: 0.001 seconds decryption: 0.0 seconds
AES (CFB)	encryption: 0,002 seconds decryption: 0,00212 seconds	encryption: 0.001 seconds decryption: 0.001 seconds
Salsa20	encryption: 0,00168 seconds decryption: 0,00112 seconds	encryption: 0.0 seconds decryption: 0.0 seconds
SHA256	0,00236 seconds	0.0 seconds

Найпростішим для використання був Pycryptodome, але й самим повільним, оскільки для тексту довжни 32 байти маємо такі показники:

Бібліотека	Pycrypto
AES (EAX)	encryption: 0.06017446517944336 seconds decryption: 0.01000213623046875 seconds
AES (CFB)	encryption: 0.09994697570800781 seconds decryption: 0.00999402999877929 seconds
Salsa20	encryption: 0.08995413780212402 seconds decryption: 0.010007143020629883 seconds

Генерація ключів в Pycrypto, де розмір модуля RSA 1024 біти зайняла 3.52 секунди, в той час як у генерація ключів де модуль RSA має розмір 2048 біт зайняла 0.053 секунди. Бібліотеки Bouncy Castle та Crypto++ показали приблизно однаковий час роботи обраних криптографічних примітивів: AES (EAX), AES (CFB), Salsa20, SHA256, але у розібратися у використанні зайняло більше часу.