

Національний технічний університет України
«Київський політехнічний інститут ім. Ігоря Сікорського»
Навчально-науковий фізико-технічний інститут

Комп'ютерний практикум №1
з курсу «Методи реалізації криптографічних
механізмів»

Тема: «Вибір та реалізація базових фреймворків та
бібліотек»

Виконав студент
групи ФІ-22мн
Максименко М.С.

Перевірила:
Байденко П. В.

Мета

Вибір базових бібліотек/сервісів для подальшої реалізації криптосистем

Завдання

Вибір бібліотеки реалізації основних криптографічних примітивів з точки зору їх ефективності за часом та пам'яттю для різних програмних платформ.

Підгрупа 2В. Порівняння бібліотек OpenSSL, crypto++, CryptoLib, PyCrypto для розробки гібридної криптосистеми під Linux платформу.

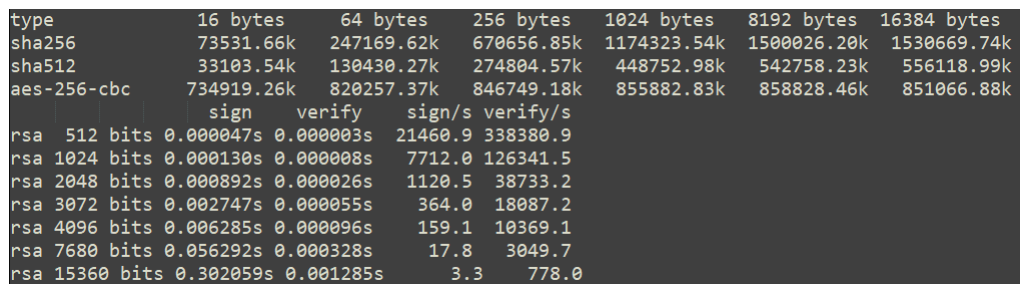
Хід роботи

OpenSSL – це криптографічна бібліотека з відкритим вихідним кодом на мові C. Це надзвичайно популярна і широко використовувана бібліотека, яка використовується для забезпечення безпечного спілкування через Інтернет. OpenSSL надає повний набір криптографічних інструментів і бібліотек, які можна використовувати для реалізації широкого спектру протоколів і алгоритмів безпеки. OpenSSL також надає утиліту командного рядка, яку можна використовувати для управління сертифікатами, ключами та для інших операцій, пов'язаними з безпекою. OpenSSL є проєктом з відкритим вихідним кодом і підтримується великою спільнотою волонтерів і розробників. OpenSSL часто оновлюється та містить виправлення помилок, щоб гарантувати, що вона залишається безпечною та надійною бібліотекою.

Утиліта OpenSSL містить команду *speed*, яка проводить бенчмарк усіх реалізованих криптографічних алгоритмів на системі користувача. Для алгоритмів, що порівнюються в цій роботі, виклик утиліти здійснювався так:

```
1 openssl speed sha256 sha512 aes-256-cbc rsa > results.txt
```

Результати зберігаються у файл results.txt (рис. 1).



type	16 bytes	64 bytes	256 bytes	1024 bytes	8192 bytes	16384 bytes
sha256	73531.66k	247169.62k	670656.85k	1174323.54k	1500026.20k	1530669.74k
sha512	33103.54k	130430.27k	274804.57k	448752.98k	542758.23k	556118.99k
aes-256-cbc	734919.26k	820257.37k	846749.18k	855882.83k	858828.46k	851066.88k
sign verify sign/s verify/s						
rsa 512 bits	0.000047s	0.000003s	21460.9	338380.9		
rsa 1024 bits	0.000130s	0.000008s	7712.0	126341.5		
rsa 2048 bits	0.000892s	0.000026s	1120.5	38733.2		
rsa 3072 bits	0.002747s	0.000055s	364.0	18087.2		
rsa 4096 bits	0.006285s	0.000096s	159.1	10369.1		
rsa 7680 bits	0.056292s	0.000328s	17.8	3049.7		
rsa 15360 bits	0.302059s	0.001285s	3.3	778.0		

Рисунок 1 – Уривок файлу results.txt

Crypto++ – це безкоштовна бібліотека C++ з відкритим вихідним кодом, яка надає криптографічні алгоритми і протоколи, включаючи шифрування, гешування, цифрові підписи, узгодження ключів, генерацію

псевдовипадкових чисел і багато іншого. Вона широко використовується в комерційних і відкритих програмах, включаючи веб-браузери, операційні системи та вбудовані пристрої.

Crypto++ була вперше випущена в 1995 році і зараз є однією з найпопулярніших бібліотек для криптографії. Бібліотека добре задокументована і має активну спільноту розробників і користувачів. Вона підтримує широкий спектр платформ, включаючи Windows, Linux, macOS, iOS та Android.

Бібліотека добре оптимізована і забезпечує реалізацію різноманітних алгоритмів, включаючи AES, DES, RC4 і SHA. Вона також включає реалізацію протоколу обміну ключами Діффі-Хеллмана на основі еліптичної кривої та алгоритму цифрового підпису на основі еліптичної кривої. Crypto++ також забезпечує підтримку захищених протоколів зв'язку, таких як SSL/TLS і SSH.

Crypto++ проста у використанні, з інтуїтивно зрозумілим API, що дозволяє легко інтегрувати криптографію в додатки. Вона також має високий рівень безпеки завдяки суворому процесу аудиту безпеки, який гарантує, що код не містить вразливостей.

Crypto++ також надає інструмент для бенчмаркінгу алгоритмів. На жаль, вбудований бенчмарк тестує всі алгоритми тільки на блоках довжини 2048 байт. Також ця утиліта не дозволяє окремо обрати бажані алгоритми, а тому тестувалися усі, а виклик здійснювався так:

```
1 cryptest b 3 3.2 > results.html
```

Отримали html-сторінку (рис. 2), яка містить результати тестування.

Crypto++ 8.7.0 Benchmarks

Here are speed benchmarks for some commonly used cryptographic algorithms.

CPU frequency of the test platform is 3.2 GHz.

Algorithm	Provider	MiB/Second	Cycles/Byte
NonblockingRng	C++	66	46.5
AutoSeededRandomPool	C++	393	7.77
AutoSeededX917RNG(AES)	AESNI	46	67.0
MT19937	C++	855	3.57
RDRAND	RDRAND	11	271.7
RDSEED	RDSEED	11	269.6
AES/OFB RNG	AESNI	864	3.53
Hash_DRBG(SHA1)	SHANI	245	12.43
Hash_DRBG(SHA256)	SHANI	341	8.94
HMAC_DRBG(SHA1)	SHANI	67	45.3
HMAC_DRBG(SHA256)	SHANI	97	31.4
CRC32	C++	415	7.35
CRC32C	SSE4.2	3614	0.84
Adler32	C++	2016	1.51
MD5	C++	545	5.60
SHA-1	SHANI	1534	1.99
SHA-256	SHANI	1480	2.06
SHA-512	SSE2	344	8.88
SHA3-224	C++	314	9.72
SHA3-256	C++	296	10.32
SHA3-384	C++	228	13.40
SHA3-512	C++	158	19.34
Keccak-224	C++	315	9.69
Keccak-256	C++	297	10.27
Keccak-384	C++	226	13.53
Keccak-512	C++	158	19.34
Tiger	SSE2	484	6.30
Whirlpool	SSE2	173	17.69
RIPEMD-160	C++	230	13.27
RIPEMD-256	C++	216	13.88

Рисунок 2 – Частина сторінки results.html

PyCryptodome – це бібліотека Python, яка надає криптографічні примітиви та алгоритми. Це форк бібліотеки PyCrypto, який пропонує різноманітні покращення порівняно з попередником, включаючи підтримку більшої кількості алгоритмів та покращену продуктивність.

PyCryptodome дозволяє розробникам швидко і легко вбудовувати шифрування, гешування і цифрові підписи в свої додатки. Вона підтримує широкий спектр алгоритмів, включаючи AES, RSA, SHA-2, SHA-3, ECC та інші. Крім того, PyCryptodome також надає інструменти для управління ключами.

PyCryptodome – це потужна бібліотека, яку можуть використовувати як досвідчені розробники, так і початківці. Її API простий та інтуїтивно зрозумілий, що полегшує початок роботи. Вона також надає детальну документацію та ряд навчальних посібників, які допоможуть користувачам

швидко набрати швидкість. Для тих, хто шукає потужну і просту у використанні бібліотеку криптографії для Python, PyCryptodome є ідеальним вибором.

Для тестування швидкодії обраних алгоритмів була написана проста утиліта за аналогом програм тестування, які доступні для попередніх алгоритмів. Аналогічно кожен із алгоритмів виконується певну задану кількість секунд на встановленому розмірі даних. По закінченню результати швидкодії обчислюються за відомою кількістю виконаних ітерацій. Код програми наступний:

```
1 import Crypto.Random
2 import time
3 from Crypto.Cipher import AES
4 from Crypto.Hash import SHA512, SHA256
5 from Crypto.PublicKey import RSA
6 from Crypto.Signature import pkcs1_15
7
8
9 def test_aes(seconds=3, block_size=2048):
10     print(f"-----AES-CBC-256 testing for {seconds} seconds.
11         -----")
12
13     key = b'doesnt matter very much only len'
14     cipher = AES.new(key, AES.MODE_CBC)
15     pt = Crypto.Random.get_random_bytes(block_size)
16
17     t_end = time.time() + seconds
18
19     iterations = 0
20     while time.time() < t_end:
21         ct = cipher.encrypt(pt)
22         iterations += 1
23
24     print(f"{iterations} = }, {block_size} bytes")
25     print(f"totaling to {iterations * block_size / 1000 / 1000 / seconds} MB/s ")
26
27 def test_sha256(seconds=3, block_size=2048):
```

```

28     print(f"-----SHA-256 testing for {seconds} seconds.
-----")
29     data = Crypto.Random.get_random_bytes(block_size)
30
31     t_end = time.time() + seconds
32     iterations = 0
33     while time.time() < t_end:
34         hasher = SHA256.new(data=data)
35         hashed = hasher.digest()
36         iterations += 1
37
38     print(f"{iterations = }, {block_size = } bytes")
39     print(f"totaling to {iterations * block_size / 1000 / 1000 / seconds} MB/s ")
40
41 def test_sha512(seconds=3, block_size=2048):
42     print(f"-----SHA-512 testing for {seconds} seconds.
-----")
43     data = Crypto.Random.get_random_bytes(block_size)
44
45     t_end = time.time() + seconds
46     iterations = 0
47     while time.time() < t_end:
48         hasher = SHA512.new(data=data)
49         hashed = hasher.digest()
50         iterations += 1
51
52     print(f"{iterations = }, {block_size = } bytes")
53     print(f"totaling to {iterations * block_size / 1000 / 1000 / seconds} MB/s ")
54
55
56
57 def test_rsa_sign(seconds=10, block_size=2048):
58     print(
59         f"-----RSA {block_size} sign testing for {seconds}
seconds.-----")
60
61     key = RSA.generate(block_size)
62     message = Crypto.Random.get_random_bytes(block_size // 8)
63     h = SHA256.new(message)

```

```

64
65     t_end = time.time() + seconds
66     iterations = 0
67     signer = pkcs1_15.new(key)
68     while time.time() < t_end:
69         signature = signer.sign(h)
70         iterations += 1
71
72     print(f"{iterations = }, {block_size = } bits")
73     print(f"totaling to {iterations / seconds} signs/s ")
74     print(f"{seconds / iterations} seconds / sign ")
75     print(f"{seconds / iterations * 1000} milliseconds / sign ")
76
77
78 def test_rsa_verify(seconds=10, block_size=2048):
79     print(
80         f"-----RSA {block_size} verify testing for {seconds}
81         seconds.-----")
82
83     key = RSA.generate(block_size)
84     message = Crypto.Random.get_random_bytes(block_size // 8)
85     h = SHA256.new(message)
86
87     signature = pkcs1_15.new(key).sign(h)
88
89     verifier = pkcs1_15.new(key)
90     t_end = time.time() + seconds
91     iterations = 0
92     while time.time() < t_end:
93         verifier.verify(h, signature)
94         iterations += 1
95
96     print(f"{iterations = }, {block_size = } bits")
97     print(f"totaling to {iterations / seconds} verifications/s ")
98     print(f"{seconds / iterations} seconds / verification ")
99     print(f"{seconds / iterations * 1000} milliseconds / verification ")
100
101 if __name__ == "__main__":

```



```

102     for test in [test_aes, test_sha256, test_sha512]:
103         for block_size in [16, 64, 256, 1024, 8192, 16384]:
104             test(block_size=block_size)
105     for test in [test_rsa_sign, test_rsa_verify]:
106         for block_size in [1024, 2048, 4096]:
107             test(block_size=block_size)

```

Результати виконання програми зображено на рис. 3.

```

AES-CBC-256 testing for 3 seconds.
iterations = 1186757, block_size = 16 bytes
totaling to 6.329370666666667 MB/s
AES-CBC-256 testing for 3 seconds.
iterations = 1139086, block_size = 64 bytes
totaling to 24.300501333333333 MB/s
AES-CBC-256 testing for 3 seconds.
iterations = 957135, block_size = 256 bytes
totaling to 81.67551999999999 MB/s
AES-CBC-256 testing for 3 seconds.
iterations = 604474, block_size = 1024 bytes
totaling to 206.32712533333336 MB/s
AES-CBC-256 testing for 3 seconds.
iterations = 140296, block_size = 8192 bytes
totaling to 383.10161066666666 MB/s
AES-CBC-256 testing for 3 seconds.
iterations = 74378, block_size = 16384 bytes
totaling to 406.20305066666667 MB/s
SHA-256 testing for 3 seconds.
iterations = 365164, block_size = 16 bytes
totaling to 1.9475413333333333 MB/s
SHA-256 testing for 3 seconds.
iterations = 355079, block_size = 64 bytes
totaling to 7.575018666666668 MB/s
SHA-256 testing for 3 seconds.
iterations = 328439, block_size = 256 bytes
totaling to 28.026794666666667 MB/s
SHA-256 testing for 3 seconds.
iterations = 234502, block_size = 1024 bytes
totaling to 80.04334933333334 MB/s
SHA-256 testing for 3 seconds.
iterations = 68849, block_size = 8192 bytes
totaling to 188.00366933333336 MB/s
SHA-256 testing for 3 seconds.
iterations = 37954, block_size = 16384 bytes
totaling to 207.27944533333334 MB/s

```

Рисунок 3 – Частина виводу програми тестування PyCryptodome

Порівняння швидкодії бібліотек здійснювалося на процесорі AMD Ryzen 5 3600 3.2 GHz. Операційна система – Ubuntu 22.04.1 LTS через середовище WSL 2 (Windows 10.0.19044). Для порівняння були обрані наступні алгоритми: SHA256, SHA512, AES-256-ECB та підпис/верифікація RSA з ключами довжини 1024, 2048, 4096 біт.

Результати тестування алгоритмів SHA256, SHA512 та AES-256-CBC наведені на рис. 4. У клітинках таблиці міститься кількість даних в МБ, які алгоритм шифрує (чи гешує) за секунду. На рис. 5 наведені результати

тестування алгоритмів цифрового підпису та верифікації, дані наведено в мілісекундах на виконання однієї операції.

Важливо зазначити, що для Crypto++ тестування проводилося тільки із блоком довжини 2048 байт. Оскільки для інших бібліотек тестування з такою довжиною не проводилося, то отримані дані для Crypto++ занесені в колонки 1024 та 8192 байт для зручності порівняння. Як можна побачити, для таких (великих) довжин блоку зростання швидкодії із зростанням блоку майже зупиняється, а тому таке порівняння справедливе.

Алгоритм	Бібліотека	Розмір блоку, байт					
		16	64	256	1024	8192	16384
SHA256	OpenSSL	73.532	247.17	670.66	1174.3	1500	1530.7
	Crypto++ (*)	-	-	-	1551.9	1551.9	-
	PyCryptodome	1.948	7.575	28.027	80.043	188	207.28
SHA512	OpenSSL	33.104	130.43	274.81	448.75	542.76	556.12
	Crypto++ (*)	-	-	-	360.71	360.71	-
	PyCryptodome	2.503	9.968	35.298	104.32	252.01	279.47
AES-256-CBC	OpenSSL	734.92	820.26	846.75	855.88	858.83	851.07
	Crypto++ (*)	-	-	-	763.37	763.37	-
	PyCryptodome	6.329	24.301	81.676	206.33	383.1	406.2

Рисунок 4 – Швидкодія алгоритмів SHA256, SHA512, AES-256-CBC різних бібліотек, МБ/с

Операція	Бібліотека	Розмір блоку, байт		
		1024	2048	4096
RSA Sign	OpenSSL	0.13	0.892	6.285
	Crypto++ (*)	0.268	1.248	-
	PyCryptodome	0.932	2.339	9.268
RSA Verify	OpenSSL	0.008	0.026	0.096
	Crypto++ (*)	0.013	0.028	-
	PyCryptodome	0.326	0.582	1.16

Рисунок 5 – Швидкодія підпису та верифікації криптосистеми RSA різних бібліотек, мс/дію

ВИСНОВКИ

Реалізації алгоритмів бібліотек Crypto++ та OpenSSL є в рази швидшими за аналогічні в PyCryptodome. Це пояснюється тим, що мови C/C++ є компільованими та більш низькорівневими, аніж мова Python.

Що стосується бібліотек Crypto++ та OpenSSL, то швидшою є друга, але загалом вони демонструють схожі показники. Цікавою є перевага Crypto++ над OpenSSL при ґешуванні блоку 1024 байт SHA256. Реалізація Crypto++ була здатна використати вбудовані в процесор інструкції саме для обчислення ґешу SHA256.

Для подальшого використання була обрана бібліотека PyCryptodome, хоча вона і є значно повільнішою. Такий вибір пояснюється тим, що бібліотека є легшою в застосуванні, а відповідно швидкість розробки зростає.