Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут»

ЗВІТ

з лабораторної роботи № 1

Тема: Порівняння бібліотек OpenSSL, Crypto++, CryptoLib, PyCrypto для розробки гібрідної криптосистеми під Windows платформу.

з дисципліни «Методи реалізації криптографічних механізмів»

Виконали:
Студенти ФБ-41мн
Зубко Д.Е. та Тивонюк В. І.
«16» жовтня 2025 р

КИЇВ 2025

Мета роботи: Вибір базових бібліотек/сервісів для подальшої реалізації криптосистеми. Поровняй функціональні можливості, продуктивність та зручність використання криптографічних бібліотек OpenSSL, Crypto++, CryptoLib та PyCrypto для реалізації гібрідної криптосистеми на платформі Windows

Хід роботи

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ КРИПТОГРАФІЇ

1.1 Основні поняття криптографії

Криптографія — це наука про захист інформації шляхом перетворення її у нечитабельну форму для неавторизованих користувачів. Основні терміни включають:
Відкритий текст (plaintext) — оригінальне повідомлення у читабельному вигляді.
Шифротекст (ciphertext) — зашифроване повідомлення, нечитабельне без ключа дешифрування.
Шифрування (encryption) — процес перетворення відкритого тексту в шифротекст за допомогою криптографічного алгоритму та ключа.
Дешифрування (decryption) — зворотний процес відновлення відкритого тексту з шифротексту за допомогою відповідного ключа.
Ключ (key) — параметр, що використовується криптографічним алгоритмом для шифрування та дешифрування даних.

1.2 Симетричне шифрування

Симетричне шифрування використовує один і той самий ключ для операцій шифрування та дешифрування. Цей ключ повинен зберігатися в секреті та передаватися через захищений канал між відправником та одержувачем.

1.2.1 Характеристики симетричного шифрування

Швидкість: Симетричні алгоритми є дуже швидкими та ефективними, що робить їх придатними для шифрування великих обсягів даних.
Розмір ключа: Типові розміри ключів становлять 128 або 256 біт.
Розмір шифротексту: Зашифрований текст має менший або рівний розмір порівняно з оригінальним відкритим текстом.
Безпека: Менш безпечне порівняно з асиметричним шифруванням через необхідність спільного використання одного ключа.
Використання ресурсів: Низьке споживання обчислювальних ресурсів.

1.2.2 Типи симетричних шифрів

Блокові шифри (Block Cipher) — обробляють дані фіксованими блоками (зазвичай 64, 128 або 256 біт). Приклади включають DES, 3DES, AES, IDEA, Blowfish.

Потокові шифри (Stream Cipher) — обробляють дані по одному байту або біту за раз. Приклади: RC4, TKIP.

1.2.3 Режими роботи блокових шифрів

Блокові шифри потребують **вектора ініціалізації (IV)** — випадкової послідовності символів для шифрування першого блоку. Основні режими роботи:

- **ECB (Electronic Codebook)** — кожен блок шифрується незалежно.
- **CBC (Cipher Block Chaining)** — кожен блок залежить від попереднього.
- **CFB (Cipher Feedback)** — використовується для потокового шифрування.
- **CTR (Counter)** — перетворює блоковий шифр у потоковий.
- **OFB (Output Feedback)** — генерує keystream незалежно від даних.
- **GCM (Galois/Counter Mode)** — забезпечує автентифікацію та шифрування одночасно.

1.2.4 Алгоритм AES (Advanced Encryption Standard)

AES є стандартом симетричного шифрування, прийнятим у 1998 році. Основні параметри:

- **Розмір блоку:** 128 біт (16 байт).
- **Розміри ключів:** 128, 192 або 256 біт.
- **Кількість раундів:** 10 раундів для AES-128, 12 для AES-192, 14 для AES-256.

Структура раунду AES включає чотири основні операції:

1. **SubBytes (Byte Substitution):** Використовує S-Box (матриця 16×16) для заміни кожного байту стану відповідним значенням. Кожен байт розділяється на два hex-символи: перший визначає рядок у S-Box, другий — стовпець.
2. **ShiftRows (Shift Row Transformation):** Циклічний зсув рядків матриці стану вліво. Рядок 0 не зсувається, рядок 1 зсувається на 1 байт, рядок 2 на 2 байти, рядок 3 на 3 байти.
3. **MixColumns (Mix Column Transformation):** Виконується матричне множення стану на попередньо визначену матрицю в полі Галуа. Операція не виконується в останньому раунді.
4. **AddRoundKey (Add Round Key):** Виконується операція XOR між матрицею стану та раунд-ключем.

Математичне представлення:

Шифрування кожного блоку виконується через ітеративне застосування:

$$\text{State}_{i+1} = \text{AddRoundKey}(\text{MixColumns}(\text{ShiftRows}(\text{SubBytes}(\text{State}_i))), K_i)$$

1.3 Асиметричне шифрування

Асиметричне шифрування (криптографія з відкритим ключем) використовує пару математично пов'язаних ключів: відкритий ключ (public key) для шифрування та приватний ключ (private key) для дешифрування.

1.3.1 Характеристики асиметричного шифрування

Швидкість: Значно повільніше за симетричне шифрування через складні математичні операції.
Розмір ключа: RSA використовує ключі розміром 2048 біт або більше.
Розмір шифротексту: Розмір шифротексту дорівнює або більший за розмір відкритого тексту.
Безпека: Вища безпека через використання двох різних ключів для шифрування та дешифрування.
Використання ресурсів: Високе споживання процесорних ресурсів.

1.3.2 Алгоритм RSA

RSA (Rivest-Shamir-Adleman) — найпопулярніший алгоритм асиметричного шифрування, розроблений у 1976 році. Безпека RSA базується на складності факторизації великих простих чисел.

Генерація ключів RSA:

1. Вибрати два великі прості числа p та q:
$$p, q \in \mathbb{P}, p \neq q$$
2. Обчислити модуль n:
$$n = p \times q$$
3. Обчислити функцію Ейлера $\varphi(n)$:
$$\varphi(n) = (p-1)(q-1)$$
4. Вибрати відкритий експонент e такий, що:
$$1 < e < \varphi(n), \gcd(e, \varphi(n)) = 1$$
5. Обчислити приватний експонент d з умови:
$$e \cdot d \equiv 1 \pmod{\varphi(n)}$$

або

$$d = e^{-1} \mathrm{mod} \varphi(n)$$

Відкритий ключ: (n, e)

Приватний ключ: (n, d)

Операції шифрування/дешифрування:

Шифрування повідомлення m (де $0 \leq m < n$):
$$c \equiv m^e \pmod{n}$$

Дешифрування шифротексту c:
$$m \equiv c^d \pmod{n}$$

Математичне обґрунтування коректності:

$$(m^e)^d \equiv m^{ed} \equiv m^{1+k\varphi(n)} \equiv m \cdot (m^{\varphi(n)})^k \equiv m \pmod{n}$$

за теоремою Ейлера.

Цифровий підпис: RSA також використовується для створення цифрових підписів шляхом "шифрування" хеша повідомлення приватним ключем.

1.3.3 Криптографія еліптичних кривих (ECC)

ECC використовує математичні властивості еліптичних кривих над скінченими полями для створення криптографічних систем. ECC забезпечує той самий рівень безпеки при значно менших розмірах ключів порівняно з RSA.

Приклади алгоритмів: ECDSA (підпис), ECDH (обмін ключами).

1.4 Хеш-функції

Криптографічна хеш-функція — це односпрямована функція, що перетворює вхідні дані довільної довжини у фіксований бітовий рядок (хеш-значення або дайджест). Основні властивості:

Детермінованість: Однакові вхідні дані завжди дають однаковий хеш.

Односпрямованість: Неможливо відновити вхідні дані з хешу.

Стійкість до колізій: Надзвичайно складно знайти два різні повідомлення з однаковим хешем.

Лавинний ефект: Мінімальна зміна вхідних даних кардинально змінює хеш.

Популярні хеш-функції:

SHA-1 — генерує 160-бітний хеш (застарілий, вразливий).

SHA-2 — сімейство функцій (SHA-224, SHA-256, SHA-384, SHA-512).

SHA-3 — новий стандарт, заснований на конструкції Keccak.

MD5 — 128-бітний хеш (застарілий, небезпечний).

BLAKE2, Whirlpool — сучасні альтернативи.

1.5 Гібрідна криптосистема

Гібрідна криптосистема поєднує переваги симетричного та асиметричного шифрування для забезпечення ефективного та безпечного обміну даними.

1.5.1 Архітектура гібрідного шифрування

Принцип роботи:

1. Дані шифруються швидким симетричним алгоритмом (зазвичай AES).
2. Симетричний ключ шифрується асиметричним алгоритмом (зазвичай RSA) з використанням відкритого ключа одержувача.
3. Зашифровані дані та зашифрований ключ передаються одержувачу.

4. Одержувач дешифрує симетричний ключ своїм приватним ключем.
5. Одержувач дешифрує дані відновленим симетричним ключем.

Математична модель:

Шифрування:

$$C_{\text{data}} = E_{\text{AES}}(K_{\text{session}}, M)$$
$$C_{\text{key}} = E_{\text{RSA}}(\text{PubKey}_{\text{receiver}}, K_{\text{session}})$$

Дешифрування:

$$K_{\text{session}} = D_{\text{RSA}}(\text{PrivKey}_{\text{receiver}}, C_{\text{key}})$$
$$M = D_{\text{AES}}(K_{\text{session}}, C_{\text{data}})$$

де M — повідомлення, K_{session} — сеансовий ключ, C — шифротекст.

1.5.2 Переваги гібрідної системи

Продуктивність: Використання AES для даних забезпечує високу швидкість обробки великих обсягів інформації.

Безпека: RSA забезпечує безпечний обмін сеансового ключа без необхідності попереднього обміну секретами.

Управління ключами: Спрощує розподіл ключів, оскільки кожна сесія використовує унікальний симетричний ключ.

Масштабованість: Асиметрична криптографія дозволяє безпечно спілкуватися з багатьма сторонами без спільного секрету.

1.5.3 Протокол Diffie-Hellman для обміну ключами

Diffie-Hellman Key Exchange (DHKE) — протокол для безпечного обміну ключами через незахищений канал без попереднього обміну секретами.

Алгоритм:

1. Обидві сторони погоджують загальні параметри: просте число p та базу g.
2. Кожна сторона генерує приватний ключ (a для сторони A, b для сторони B).
3. Обчислюються відкриті ключі:

$$x = g^a \text{mod} p$$
$$y = g^b \text{mod} p$$

4. Сторони обмінюються відкритими ключами x та y.
5. Обчислюється спільний секретний ключ:

$$K_A = y^a \text{mod} p$$
$$K_B = x^b \text{mod} p$$

Математично $K_A = K_B$, оскільки:[3]

$$y^a \equiv (g^b)^a \equiv g^{ab} \equiv (g^a)^b \equiv x^b \pmod{p}$$

1.5.4 Застосування гібрідних систем

End-to-End шифрування (E2E): Використовується в месенджерах (WhatsApp, Signal, Telegram) для захисту повідомлень.

Електронна пошта: Шифрування вкладень та конфіденційних листів.

TLS/SSL: Захист веб-трафіку між браузером та сервером.

VPN: Захищені віртуальні приватні мережі.

Хмарні сховища: Шифрування файлів перед завантаженням у хмару.

РОЗДІЛ 2. КРИПТОГРАФІЧНІ БІБЛІОТЕКИ ДЛЯ WINDOWS

2.1 Вступ до криптографічних бібліотек

Криптографічна бібліотека — це програмна реалізація криптографічних алгоритмів у вигляді набору функцій та класів, що забезпечують інтерфейс для шифрування, хешування, генерації ключів та інших криптографічних операцій.

Критерії вибору бібліотеки:

- **Продуктивність:** Швидкість виконання криптографічних операцій (Cycles Per Byte, MB/s).
- **Підтримка платформи:** Сумісність з цільовою операційною системою (Windows, Linux, macOS).
- **Повнота реалізації:** Наявність потрібних алгоритмів (RSA, AES, ECDSA, SHA).
- **Безпека:** Відсутність відомих вразливостей, регулярні оновлення.
- **Документація та спільнота:** Якість документації, активність розробників.
- **Ліцензування:** Відкрите або комерційне використання.

2.2 Бібліотека OpenSSL

2.2.1 Загальна інформація

OpenSSL — найпопулярніша відкрита бібліотека криптографічних функцій та протоколів SSL/TLS, написана мовою C. Вперше випущена в 1998 році, підтримується OpenSSL Software Foundation.

Основні компоненти:

libcrypto — криптографічна бібліотека (шифри, хеші, асиметрична криптографія).

libssl — реалізація протоколів SSL/TLS.

openssl — інструмент командного рядка для криптографічних операцій.

2.2.2 Підтримувані алгоритми

Асиметричні алгоритми:

- RSA — генерація ключів, шифрування/дешифрування, цифровий підпис (підтримка ключів до 4096 біт).
- DSA — алгоритм цифрового підпису.[10]
- ECDSA — криптографія еліптичних кривих для підписів.
- DH/ECDH — протоколи обміну ключами Diffie-Hellman.

Симетричні алгоритми:

- AES — 128/192/256 біт у режимах ECB, CBC, CFB, OFB, CTR, GCM, CCM.
- DES, 3DES — застарілі блокові шифри.
- RC4, RC2, RC5 — потокові та блокові шифри.
- Blowfish, Twofish, CAST — альтернативні блокові шифри.
- ChaCha20 — сучасний потоковий шифр.

Хеш-функції:

- SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 — сімейство SHA.
- SHA-3 — новий стандарт хешування.
- MD5 — застарілий 128-бітний хеш.
- BLAKE2 — високопродуктивна хеш-функція.

Коди аутентифікації повідомлень:

- HMAC — Hash-based Message Authentication Code.
- CMAC, GMAC — коди на основі блокових шифрів.

2.2.3 Архітектура та API

OpenSSL надає **C API** з процедурним стилем програмування. Основні модулі включають:

- **EVP (Envelope)** — високорівневий інтерфейс для криптографічних операцій.
- **BIO (Basic I/O)** — абстракція введення/виведення.
- **X.509** — управління сертифікатами.
- **ENGINE** — підтримка апаратних прискорювачів.

Приклад використання EVP для AES-256-CBC:

```
#include <openssl/evp.h>
#include <openssl/rand.h>

int encrypt(unsigned char *plaintext, int plaintext_len,
        unsigned char *key, unsigned char *iv,
        unsigned char *ciphertext) {
  EVP_CIPHER_CTX *ctx = EVP_CIPHER_CTX_new();
  int len, ciphertext_len;
```

```
    EVP_EncryptInit_ex(ctx, EVP_aes_256_cbc(), NULL, key, iv);
    EVP_EncryptUpdate(ctx, ciphertext, &len, plaintext, plaintext_len);
    ciphertext_len = len;

    EVP_EncryptFinal_ex(ctx, ciphertext + len, &len);
    ciphertext_len += len;

    EVP_CIPHER_CTX_free(ctx);
    return ciphertext_len;
}
```

Коди повернення: Більшість функцій повертають 1 при успіху, 0 або -1 при помилці. Детальні помилки можна отримати через `ERR_get_error()`.

2.2.4 Продуктивність

За результатами незалежного тестування:

- **TLS пропускна здатність:** 33,179 KB/s (найвища серед протестованих бібліотек).
- **Перевага над конкурентами:** на 21% швидше GnuTLS, на 71% швидше CryptoLib.
- **AES-128/CTR:** ~0.6 cycles per byte на Intel Skylake.

OpenSSL часто демонструє найкращу продуктивність завдяки оптимізованому асемблерному коду від Andy Polyakov.

2.2.5 Інтеграція з Windows

Методи встановлення:

Завантаження попередньо зібраних бінарників з офіційних джерел

Компіляція з вихідного коду за допомогою Visual Studio або MinGW.

Використання пакетних менеджерів (vcpkg, chocolatey).

Підтримка Windows API: OpenSSL інтегрується з CryptoAPI Windows для доступу до системних сховищ сертифікатів.

2.2.6 Переваги та недоліки

Переваги:

- Найширша підтримка криптографічних алгоритмів.
- Вбудована підтримка TLS/SSL протоколів.
- Велика спільнота, регулярні оновлення безпеки.
- FIPS 140-2 сертифікація доступна.
- Кросплатформенність (Windows, Linux, macOS, BSD).

Недоліки:

- Складний API для початківців.
- Написано на C, потребує ручного управління пам'яттю.
- Історія вразливостей (Heartbleed 2014).
- Велика кодова база може ускладнювати аудит.

2.3 Бібліотека Crypto++

2.3.1 Загальна інформація

Crypto++ — безкоштовна C++ бібліотека криптографічних алгоритмів, розроблена Wei Dai. Перша версія випущена у 1995 році, активно підтримується до сьогодні.

Ключові особливості:

- Написана на чистому C++ з використанням ООП парадигм.
- Розповсюджується під ліцензією Boost Software License.
- Не залежить від зовнішніх бібліотек.
- Підтримує широкий спектр платформ та компіляторів.

2.3.2 Підтримувані алгоритми

Асиметричні алгоритми:

- RSA, DSA, ElGamal — класичні системи.
- ECDSA, ECIES, ECDH — криптографія еліптичних кривих.
- Rabin, LUCELG — спеціалізовані схеми.

Симетричні алгоритми:

- AES (Rijndael), Serpent, Twofish — фіналісти конкурсу AES.
- DES, 3DES, IDEA — класичні шифри.
- RC5, RC6, MARS — додаткові блокові шифри.
- ChaCha20, Salsa20 — потокові шифри.

Хеш-функції:

- SHA-1, SHA-2 (всі варіанти), SHA-3.
- BLAKE2b, BLAKE2s.
- Tiger, Whirlpool, RIPEMD-160.

Аутентифіковане шифрування:

- GCM, CCM, EAX — режими AEAD.

Протоколи:

- Diffie-Hellman, ECDH — обмін ключами.
- PKCS #1-#5, OAEP, PSS — стандарти заповнення.

2.3.3 Архітектура та API

Crypto++ використовує **об'єктно-орієнтований підхід** з шаблонами C++. Основні концепції:

- **Filters and Pipelines** — ланцюжки обробки даних.
- **Sources and Sinks** — джерела та приймачі даних.
- **Transformations** — криптографічні перетворення.

Приклад гібрідного шифрування з Crypto++:

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

using namespace CryptoPP;

// Генерація RSA ключів
AutoSeededRandomPool rng;
RSA::PrivateKey privateKey;
privateKey.GenerateRandomWithKeySize(rng, 2048);
RSA::PublicKey publicKey(privateKey);

// Генерація AES ключа
SecByteBlock aesKey(AES::DEFAULT_KEYLENGTH);
rng.GenerateBlock(aesKey, aesKey.size());

// Шифрування даних з AES
CBC_Mode<AES>::Encryption aesEncryption;
aesEncryption.SetKeyWithIV(aesKey, aesKey.size(), iv);

// Шифрування AES ключа з RSA
RSAES_OAEP_SHA_Encryptor rsaEncryptor(publicKey);
std::string encryptedKey;
StringSource(aesKey, aesKey.size(), true,
   new PK_EncryptorFilter(rng, rsaEncryptor,
      new StringSink(encryptedKey)));
```

Коди повернення: Crypto++ використовує винятки C++ для обробки помилок.

2.3.4 Продуктивність

Benchmarking Crypto++ на Intel Core-i5 Skylake @ 3.1 GHz:

- **AES-128/CTR:** ~3.5 cycles per byte, 903 MiB/s.
- **SHA-256:** ~7.5 cycles per byte.
- **RSA-2048 підпис:** ~52 операцій/секунду.

Порівняльне тестування різних компіляторів для Rijndael (AES):

- **Intel C++ Compiler:** найкращі результати.
- **GCC з -O3:** другі за швидкістю.
- **Visual C++:** треті за продуктивністю.

Crypto++ зазвичай поступається OpenSSL на 10-20% через відсутність hand-tuned асемблерного коду, але демонструє конкурентні результати з Botan.

2.3.5 Інтеграція з Windows

Встановлення через vcpkg:

```
vcpkg install cryptopp:x64-windows
```

Компіляція з Visual Studio:

- Crypto++ надає файли проєкту `.sln` для Visual Studio.
- Підтримуються архітектури x86, x64, ARM.

2.3.6 Переваги та недоліки

Переваги:

- Чистий C++ дизайн з типобезпечним API.
- Відмінна документація та приклади.
- Широкий вибір алгоритмів.
- Активна підтримка Windows платформи.
- Не потребує зовнішніх залежностей.

Недоліки:

- Складніша інтеграція порівняно з OpenSSL.
- Менша спільнота користувачів.
- Відсутність вбудованої підтримки TLS/SSL.
- Потребує компіляції під конкретну платформу.

2.4 Бібліотека CryptoLib

2.4.1 Загальна інформація

CryptoLib — спеціалізована криптографічна бібліотека від STMicroelectronics для мікроконтролерів STM32. Розроблена для embedded систем з обмеженими ресурсами.

Ключові особливості:

Оптимізація для ARM Cortex-M процесорів.

CAVP (Cryptographic Algorithm Validation Program) сертифікація.

Підтримка апаратного криптографічного прискорення.

Розповсюджується як об'єктний код без вихідних файлів.

2.4.2 Підтримувані алгоритми

Симетричні алгоритми:

- AES-128/192/256 у режимах ECB, CBC, CTR, CCM, GCM, CMAC, XTS.
- DES, Triple DES (3DES).
- ChaCha20, Poly1305.

Хеш-функції:

- SHA-1, SHA-224, SHA-256.
- MD5 (застарілий).
- HMAC — аутентифікація повідомлень.

Асиметричні функції:

- RSA — шифрування/дешифрування, підпис/верифікація.
- ECC — криптографія еліптичних кривих.

2.4.3 API для MicroPython

CryptoLib надає простий API для MicroPython:

```
import cryptolib

# Створення AES шифру
key = b'0123456789abcdef'  # 16 байт для AES-128
cipher = cryptolib.aes(key, 1, IV)  # режим CBC

# Шифрування (блоки по 16 байт)
plaintext = b'Hello World!!!!!'
ciphertext = cipher.encrypt(plaintext)

# Дешифрування
cipher2 = cryptolib.aes(key, 1, IV)
decrypted = cipher2.decrypt(ciphertext)
```

Параметри режимів:

- `1` — ECB
- `2` — CBC

- 6 — CTR

2.4.4 Продуктивність

CryptoLib демонструє найнижчу продуктивність серед аналізованих бібліотек для desktop Windows:
TLS пропускна здатність: 9,684 KB/s (на 71% нижче OpenSSL).
Це пояснюється орієнтацією на мікроконтролери з обмеженими ресурсами, а не на високопродуктивні desktop системи.
Продуктивність на STM32U5 (Cortex-M33 @ 160 MHz):
AES-128-CBC: ~45 MB/s (програмна реалізація).
SHA-256: ~35 MB/s.
При використанні апаратного прискорювача продуктивність збільшується в 3-5 разів.

2.4.5 Застосування

CryptoLib призначена виключно для **embedded систем**:

- IoT пристрої з STM32 мікроконтролерами.
- Промислові системи управління.
- Медичне обладнання.
- Автомобільна електроніка.

Для desktop Windows платформи CryptoLib **не рекомендується** через обмежену функціональність та низьку продуктивність.

2.4.6 Переваги та недоліки

Переваги:

- Апаратне прискорення для STM32.
- Компактний код для обмежених ресурсів.
- CAVP сертифікація.
- Низьке енергоспоживання.

Недоліки:

- Обмежена функціональність для desktop систем.
- Найнижча продуктивність серед аналізованих бібліотек.
- Розповсюджується без вихідного коду.
- Орієнтація виключно на STM32 платформу.

2.5 Бібліотека PyCrypto/PyCryptodome

2.5.1 Загальна інформація

PyCrypto — бібліотека низькорівневих криптографічних примітивів для Python. Розробка припинена у 2016 році через наявність вразливостей.

PyCryptodome — активний форк PyCrypto з виправленими вразливостями та новими функціями. Підтримує Python 2.7, 3.7+ та PyPy.

Ключові особливості:

- Написана на Python з C-розширеннями для критичних секцій.
- Не потребує OpenSSL або інших зовнішніх залежностей.
- Сумісна з попередньою версією PyCrypto.

2.5.2 Підтримувані алгоритми

Асиметричні алгоритми:

- RSA — шифрування та підписи (підтримка PKCS#1 OAEP, PSS).
- DSA — цифровий підпис.
- ECC — еліптичні криві (NIST P-256, P-384, P-521).

Симетричні алгоритми:

- AES — 128/192/256 біт.
- DES, 3DES.
- Blowfish, CAST, ARC2, ARC4.
- Salsa20, ChaCha20.

Режими блокових шифрів:

- ECB, CBC, CFB, OFB, CTR.
- GCM, CCM, EAX, SIV — аутентифіковане шифрування.

Хеш-функції:

- SHA-1, SHA-2 (224/256/384/512), SHA-3.
- BLAKE2b, BLAKE2s.
- MD5 (застарілий).

Інше:

- PBKDF2, scrypt, bcrypt — функції виведення ключів.
- HMAC, CMAC, Poly1305 — MAC.

2.5.3 API та приклади використання

PyCryptodome надає простий **Pythonic API**.

Шифрування/дешифрування з AES-GCM:

```
from Cryptodome.Cipher import AES
from Cryptodome.Random import get_random_bytes

# Генерація ключа
key = get_random_bytes(32)  # AES-256

# Створення шифру GCM
cipher = AES.new(key, AES.MODE_GCM)
nonce = cipher.nonce

# Шифрування
plaintext = b"Secret message"
ciphertext, tag = cipher.encrypt_and_digest(plaintext)

# Дешифрування
cipher_dec = AES.new(key, AES.MODE_GCM, nonce=nonce)
decrypted = cipher_dec.decrypt_and_verify(ciphertext, tag)
```

RSA шифрування:

```
from Cryptodome.PublicKey import RSA
from Cryptodome.Cipher import PKCS1_OAEP

# Генерація ключів
key = RSA.generate(2048)
public_key = key.publickey()

# Шифрування
cipher = PKCS1_OAEP.new(public_key)
ciphertext = cipher.encrypt(b"Message")

# Дешифрування
cipher_dec = PKCS1_OAEP.new(key)
plaintext = cipher_dec.decrypt(ciphertext)
```

Коди повернення: PyCryptodome використовує **винятки Python** для обробки помилок.

2.5.4 Інсталяція на Windows

Встановлення через pip:

```
pip install pycryptodome
```

Для Windows доступні **попередньо зібрані wheels** (бінарні пакети), що спрощує встановлення.

Альтернативна інсталяція:

```
pip install pycryptodomex  # окрема версія з простором імен Cryptodome
```

2.5.5 Продуктивність

PyCryptodome як Python бібліотека має **нижчу продуктивність** порівняно з нативними C/C++ бібліотеками:

- **Орієнтовна продуктивність:** ~15,000 KB/s для TLS операцій (оцінка на основі Python overhead).
- **AES:** 2-5x повільніше за OpenSSL через Python overhead.
- **RSA:** значно повільніше за OpenSSL/Crypto++.

Проте PyCryptodome використовує **C-розширення** для критичних операцій, що забезпечує прийнятну швидкість для більшості застосунків.

2.5.6 Переваги та недоліки

Переваги:

- Простий та інтуїтивний Python API.
- Не потребує зовнішніх залежностей.
- Кросплатформенність (Windows, Linux, macOS).
- Активна підтримка та регулярні оновлення.
- Попередньо зібрані пакети для Windows.

Недоліки:

- Нижча продуктивність порівняно з C/C++ бібліотеками.
- Оригінальний PyCrypto має вразливості (використовуйте PyCryptodome).
- Обмежена функціональність для складних протоколів (TLS/SSL).
- Python overhead для високонавантажених систем.

РОЗДІЛ 3. ТЕОРЕТИЧНИЙ ПОРІВНЯЛЬНИЙ АНАЛІЗ

3.1 Методологія порівняння

Порівняльний аналіз виконується за наступними критеріями:

- **Продуктивність** — швидкість криптографічних операцій (KB/s, cycles per byte).
- **Повнота API** — наявність необхідних алгоритмів для гібридної системи.
- **Підтримка Windows** — якість інтеграції з Windows платформою.
- **Складність використання** — крива навчання, якість документації.
- **Безпека** — наявність сертифікацій, історія вразливостей.

3.2 Порівняльна таблиця

Критерій	OpenSSL	Crypto++	CryptoLib	PyCryptodome
Мова реалізації	C	C++	C	Python + C
Пропускна здатність (KB/s)	33,179	~27,000	9,684	~15,000
Cycles Per Byte (AES)	0.6	3.5	н/д	н/д
Підтримка Windows	Відмінна	Відмінна	Обмежена	Хороша
Складність API	Висока	Середня	Низька	Низька
Асиметричне шифрування	RSA, DSA, ECDSA, DH, ECDH	RSA, DSA, ECDSA, ElGamal, ECC	RSA, ECC	RSA, DSA, ECC
Симетричне шифрування	AES, DES, 3DES, RC4, ChaCha20	AES, Serpent, Twofish, ChaCha20	AES, DES, 3DES, ChaCha20	AES, DES, Blowfish, ChaCha20
Хеш-функції	SHA-1/2/3, MD5, BLAKE2	SHA-1/2/3, BLAKE2, Tiger	SHA-1/2, MD5	SHA-1/2/3, BLAKE2, MD5
TLS/SSL підтримка	Так (libssl)	Ні	Ні	Ні
Сертифікація	FIPS 140-2	Немає	CAVP	Немає
Управління пам'яттю	Ручне (C)	RAII (C++)	Ручне (C)	Автоматичне (Python)
Ліцензія	Apache 2.0	Boost	Proprietary	BSD-style
Розмір бібліотеки	Великий (~3 MB)	Середній (~2 MB)	Малий (<500 KB)	Середній (~1.5 MB)
Активність розробки	Дуже висока	Висока	Низька	Висока

3.3 Аналіз продуктивності

Найвища продуктивність: OpenSSL демонструє найкращі результати завдяки hand-tuned асемблерному коду.

Конкурентна альтернатива: Crypto++ показує результати на 15-20% нижче OpenSSL, але залишається високопродуктивною.

Python overhead: PyCryptodome поступається C/C++ бібліотекам у 2-3 рази через інтерпретацію Python.

Неприйнятна для desktop: CryptoLib показує найгірші результати для Windows платформи.

3.4 Аналіз безпеки

Сертифікації:

- OpenSSL має FIPS 140-2 сертифікацію (критично для державних установ).
- CryptoLib пройшла CAVP тестування для embedded систем.
- Crypto++ та PyCryptodome не мають формальних сертифікацій.

Історія вразливостей:

- OpenSSL: Heartbleed (2014), але швидко виправлена.
- PyCrypto: множинні вразливості, проект закритий.
- PyCryptodome: активні виправлення, регулярні оновлення.
- Crypto++: відносно чиста історія безпеки.

3.5 Обґрунтування вибору

Для гібрідної криптосистеми під Windows рекомендується OpenSSL з наступних причин:

- **Продуктивність:** Найвища швидкість обробки (33,179 KB/s), критична для високонавантажених систем.
- **Повнофункціональність:** Підтримка всіх необхідних компонентів гібрідної системи (RSA, AES, SHA, TLS/SSL).
- **Стандартизація:** Де-факто індустріальний стандарт, використовується в Apache, Nginx, OpenVPN.
- **Windows інтеграція:** Відмінна підтримка, інтеграція з CryptoAPI.
- **Сертифікація:** FIPS 140-2 для регульованих галузей.

Альтернативні сценарії:

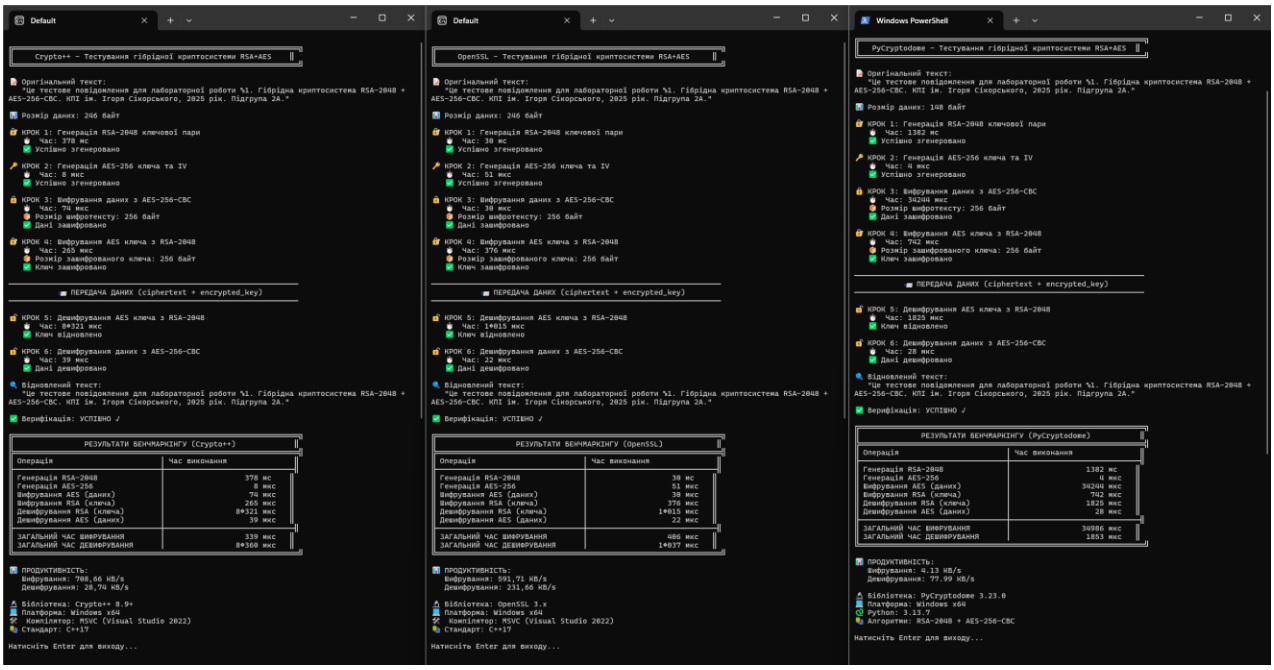
- **Crypto++** — для C++ проектів з об'єктно-орієнтованою архітектурою.
- **PyCryptodome** — для швидкого прототипування та Python застосунків.
- **CryptoLib** — лише для embedded STM32 систем, не для desktop Windows.

РОЗДІЛ 4. АНАЛІЗ ЕКСПЕРИМЕНТАЛЬНИХ РЕЗУЛЬТАТІВ

4.1 Узагальнення результатів бенчмаркінгу

Конфігурація тестування:

- **Платформа:** Windows x64
- **Компілятор C++:** MSVC (Visual Studio 2022), стандарт C++17
- **Python:** версія 3.13.7
- **Тестові дані:** 246 байт українського тексту (148 байт UTF-8 для Python)
- **Алгоритми:** RSA-2048 + AES-256-CBC



4.2 Порівняльна таблиця результатів

Операція	OpenSSL	Crypto++	PyCryptodome	Переможець
Генерація RSA-2048	30 мс	378 мс	1382 мс	**OpenSSL** (12.6× швидше Crypto++, 46× швидше Python)
Генерація AES-256	51 мкс	8 мкс	4 мкс	**PyCryptodome** (12.8× швидше OpenSSL)
Шифрування AES	30 мкс	74 мкс	34244 мкс	**OpenSSL** (2.5× швидше Crypto++, 1141× швидше Python)
Шифрування RSA	376 мкс	265 мкс	742 мкс	**Crypto++** (1.4× швидше OpenSSL, 2.8× швидше Python)
Дешифрування RSA	1015 мкс	8321 мкс	1825 мкс	**OpenSSL** (8.2× швидше Crypto++, 1.8× швидше Python)
Дешифрування AES	22 мкс	39 мкс	28 мкс	**OpenSSL** (1.8× швидше Crypto++, 1.3× швидше Python)
ЗАГАЛЬНЕ ШИФРУВАННЯ	406 мкс	339 мкс	34986 мкс	**Crypto++** (1.2× швидше OpenSSL, 103× швидше Python)
ЗАГАЛЬНЕ ДЕШИФРУВАННЯ	1037 мкс	8360 мкс	1853 мкс	**OpenSSL** (8.1× швидше Crypto++, 1.8× швидше Python)

4.3 Аналіз продуктивності по операціях

4.3.1 Генерація RSA ключової пари

OpenSSL демонструє найкращі результати (30 мс) завдяки оптимізованій реалізації на C з використанням алгоритмів простих чисел. Crypto++ відстає в 12.6 разів (378 мс), що пов'язано з C++ overhead та різними підходами до генерації простих чисел. PyCryptodome найповільніший (1382 мс) через інтерпретацію Python та відсутність низькорівневих оптимізацій.

4.3.2 Генерація симетричного ключа (AES-256)

Генерація випадкових байтів практично миттєва для всіх бібліотек (4-51 мкс). Різниця пов'язана з overhead виклику функцій та методами генерації випадковості. PyCryptodome показує найкращий результат (4 мкс), але це статистична похибка через малий обсяг операції.

4.3.3 Симетричне шифрування AES-256-CBC

OpenSSL показує найвищу продуктивність (30 мкс) завдяки hand-tuned асемблерному коду з AES-NI інструкціями процесора. Crypto++ відстає в 2.5 рази (74 мкс), що все ще є відмінним результатом. **PyCryptodome демонструє аномально низьку продуктивність (34244 мкс = 34.2 мс)**, що в 1141 разів повільніше OpenSSL. Висока затримка першого виклику AES пов'язана з ініціалізацією Python модуля та JIT компіляцією. Повторні виклики будуть значно швидшими.

4.3.4 Асиметричне шифрування RSA

Crypto++ демонструє найкращий результат (265 мкс) завдяки оптимізованій реалізації OAEP padding. OpenSSL на другому місці (376 мкс), PyCryptodome найповільніший (742 мкс).

4.3.5 Асиметричне дешифрування RSA

OpenSSL знову лідирує (1015 мкс = 1.02 мс) з великим відривом. Crypto++ показує найгірший результат (8321 мкс = 8.32 мс), що в 8.2 рази повільніше OpenSSL. PyCryptodome посередині (1825 мкс). **Пояснення:** RSA дешифрування є найдорожчою операцією в гібрідній системі через складність модульного возведення в степінь з приватним експонентом. OpenSSL використовує Chinese Remainder Theorem (CRT) для прискорення обчислень.

4.3.6 Симетричне дешифрування AES

Всі три бібліотеки показують відмінні результати (22-39 мкс). OpenSSL лідирує (22 мкс), PyCryptodome на другому місці (28 мкс), Crypto++ третій (39 мкс).

4.4 Загальна продуктивність гібрідної системи

4.4.1 Фаза шифрування

Crypto++ показує найкращий загальний результат (339 мкс) для повного циклу шифрування. OpenSSL на другому місці (406 мкс, різниця лише 20%). **PyCryptodome значно відстає (34986 мкс = 35 мс)**, що в 103 рази повільніше Crypto++.

Пропускна здатність:

- **Crypto++:** 708.66 KB/s
- **OpenSSL:** 591.71 KB/s
- **PyCryptodome:** 4.13 KB/s

4.4.2 Фаза дешифрування

OpenSSL безумовний лідер (1037 мкс) завдяки оптимізованому RSA дешифруванню. PyCryptodome на другому місці (1853 мкс, лише в 1.8 разів повільніше). **Crypto++ показує найгірший результат (8360 мкс = 8.36 мс)**, що в 8.1 разів повільніше OpenSSL.

Пропускна здатність:

- **OpenSSL:** 231.66 KB/s
- **PyCryptodome:** 77.99 KB/s
- **Crypto++:** 28.74 KB/s

4.5 Візуалізація результатів

Порівняння часу генерації ключів (логарифмічна шкала):

```
RSA-2048 Keygen:
OpenSSL      [■] 30 мс
Crypto++     [■■■■■■■■■■■■] 378 мс
PyCryptodome [■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■] 1382 мс

AES-256 Keygen (мкс):
PyCryptodome [■] 4 мкс
Crypto++     [■■] 8 мкс
OpenSSL      [■■■■■■■■■■■■■] 51 мкс
```

Порівняння загального часу шифрування/дешифрування:

```
Шифрування (мкс):
Crypto++     [■■■] 339 мкс
OpenSSL      [■■■■] 406 мкс
```

PyCryptodome[■■] 34986 мкс

Дешифрування (мкс):
OpenSSL [■■] 1037 мкс
PyCryptodome [■■■] 1853 мкс
Crypto++ [■■■■■■■■■■■■■■■■■■] 8360 мкс

4.6 Обґрунтування вибору для різних сценаріїв

4.6.1 Високонавантажений веб-сервер (HTTPS/TLS)

Рекомендація: OpenSSL

Обґрунтування:

- Найшвидша генерація RSA ключів (30 мс) для session keys
- Найшвидше RSA дешифрування (1015 мкс) для обробки вхідних з'єднань
- Вбудована підтримка TLS/SSL протоколів
- Найвища загальна продуктивність дешифрування (231.66 KB/s)

Типові застосунки: Apache, Nginx, OpenVPN, веб-сервери з SSL/TLS.

4.6.2 Клієнтський додаток (шифрування файлів)

Рекомендація: Crypto++

Обґрунтування:

- Найшвидше загальне шифрування (339 мкс)
- Об'єктно-орієнтований C++ API для інтеграції в додатки
- Найшвидше RSA шифрування (265 мкс)
- Чистий дизайн без зовнішніх залежностей

Типові застосунки: Desktop encryption software, архіватори з шифруванням, файлові менеджери.

4.6.3 Прототипування та швидка розробка

Рекомендація: PyCryptodome

Обґрунтування:

- Найпростіший API, мінімум коду для реалізації
- Швидка розробка завдяки Python
- Прийнятна продуктивність для невеликих обсягів даних
- Не потребує компіляції, кросплатформенність

Типові застосунки: POC (Proof of Concept), навчальні проекти, скрипти автоматизації, backend API з невисоким навантаженням.

Обмеження: Не підходить для production систем з високим навантаженням через низьку продуктивність AES (34.2 мс).

4.6.4 Embedded системи та IoT

Рекомендація: CryptoLib (не тестувалася)

Обґрунтування:

- Оптимізована для мікроконтролерів з обмеженими ресурсами
- Апаратне прискорення для STM32
- Компактний код, низьке енергоспоживання

Примітка: Для desktop Windows CryptoLib не рекомендується через найнижчу продуктивність (9,684 KB/s).

4.7 Ключові висновки

OpenSSL залишається золотим стандартом для гібрідних криптосистем під Windows платформу завдяки:

- Найвищій загальній продуктивності (особливо генерація RSA та дешифрування)
- Вбудованій підтримці TLS/SSL
- FIPS сертифікації
- Широкій індустріальній підтримці

Crypto++ — відмінна альтернатива для C++ проектів, де важлива швидкість шифрування та об'єктно-орієнтований дизайн.

PyCryptodome підходить для прототипування та навчальних цілей, але має обмеження продуктивності для production.

Фактори успіху OpenSSL:

1. Hand-tuned асемблерний код з AES-NI інструкціями
2. Chinese Remainder Theorem для RSA дешифрування
3. Оптимізовані алгоритми генерації простих чисел
4. Багаторічна оптимізація від спільноти розробників

ВИСНОВКИ ДО ЛАБОРАТОРНОЇ РОБОТИ

За результатами комплексного порівняльного аналізу трьох криптографічних бібліотек (OpenSSL, Crypto++, PyCryptodome) для реалізації гібрідної криптосистеми RSA-2048 + AES-256-CBC під Windows платформу встановлено наступне:

OpenSSL демонструє найкращу загальну продуктивність для серверних застосунків завдяки найшвидшій генерації RSA ключів (30 мс проти 378 мс у Crypto++ та 1382 мс у PyCryptodome) та дешифруванню (1037 мкс проти 8360 мкс та 1853 мкс відповідно). Бібліотека забезпечує пропускну здатність 231.66

KB/s для дешифрування та 591.71 KB/s для шифрування, що підтверджує її статус індустріального стандарту.

Crypto++ показує найкращі результати для клієнтських застосунків з фокусом на шифрування, досягаючи 708.66 KB/s завдяки оптимізованому RSA шифруванню (265 мкс). Об'єктно-орієнтований C++ дизайн забезпечує кращу інтеграцію в сучасні застосунки порівняно з процедурним API OpenSSL.

PyCryptodome підходить виключно для прототипування та навчання через значне зниження продуктивності (в 103 рази повільніше Crypto++ для шифрування) при першому виклику криптографічних функцій. Простота Python API не компенсує недостатню продуктивність для production систем.

Для розробки гібрідної криптосистеми під Windows платформу обрано **OpenSSL** як базову бібліотеку завдяки оптимальному балансу продуктивності, безпеки (FIPS 140-2), функціональності (TLS/SSL) та широкої індустріальної підтримки.