НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. Ігоря
СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ

Звіт з виконання лабораторної роботи

Порівняння бібліотек OpenSSL, Crypto++, CryptoLib, PyCrypto для розробки гібрідної криптосистеми під Linux платформу

Виконали студенти
групи ФІ-52мн
Геращенко Володимир
Дорошенко Юрій
Волошин Ігор

Київ — 2025

1 Мета роботи

Вибір базової бібліотеки реалізації криптографічних примітивів з точки зору їх ефективності за часом та пам'яттю для різних програмних платформ. Порівняння бібліотек OpenSSL, Crypto++, CryptoLib, PyCryptodome для розробки гібрідної криптосистеми під Linux платформу:

1. підтримуваних криптографічних примітивів (симетричних, асиметричних, гешів, MAC, KDF),
2. ефективності за часом та пам'яттю на платформі Linux,
3. функціональні можливочті та придатності для побудови гібридної криптосистеми

2 Хід роботи

2.1 Гібридна криптосистема: означення, структура та призначення

У сучасних системах інформаційної безпеки важливу роль відіграють **гібридні криптосистеми**. Гібридною називається криптосистема, що поєднує у собі переваги як симетричної, так і асиметричної криптографії. Основна мотивація такого поєднання випливає з фундаментальних властивостей цих двох класів алгоритмів.

Симетричні криптосистеми забезпечують дуже високу швидкість шифрування та лінійну часову складність $O(n)$ відносно розміру даних, але для їх використання необхідно, щоб обидві сторони заздалегідь володіли спільним секретним ключем. Натомість асиметричні криптосистеми дають можливість безпечного обміну ключами без попередньої домовленості, але мають значно більші обчислювальні накладні витрати: часові витрати ростуть пропорційно складності операцій з великими числами, що в загальному випадку є кубічними або квазіквадратичними за розміром модуля.

Гібридна криптосистема поєднує сильні сторони обох підходів:

- асиметрична криптосистема використовується лише для шифрування короткого випадкового (*сеансового*) симетричного ключа;
- симетричний алгоритм застосовується для шифрування основного масиву даних, що забезпечує високу продуктивність.

Типова гібридна криптосистема складається з таких криптографічних примітивів:

1. **Симетричний алгоритм шифрування.** Для шифрування даних використовують блочні шифри, наприклад AES із довжиною ключа 128, 192 або 256 біт. Симетричне шифрування забезпечує конфіденційність і застосовується до великих обсягів даних, оскільки має лінійну

обчислювальну складність $O(n)$. Застосовуються різні режими роботи блочних шифрів (CBC, CTR, GCM, CFB тощо), які визначають спосіб обробки блоків і характеристики стійкості.

2. **Асиметричне шифрування або протокол обміну ключами.** Короткий симетричний ключ передається за допомогою асиметричного алгоритму, наприклад RSA або еліптичних кривих (ECDH, ECIES). Асиметрична частина забезпечує захищений обмін ключами між сторонами, які до цього не мали спільного секрету. Зазвичай застосовують паддінг OAEP у RSA для забезпечення стійкості до атак із вибраним повідомленням. Завдання асиметричної компоненти — гарантувати **конфіденційність сеансового ключа та автентичність джерела** при використанні розширених схем підпису.

3. **Криптографічна геш-функція.** геш-функція використовується для отримання унікального дайджесту даних, побудови схем автентифікації повідомлень, забезпечення цілісності даних і виступає компонентом різних протоколів та паддінгів (наприклад RSA-OAEP). Поширені геш-функції: SHA-256, SHA-384, SHA-512. геш-функція має обчислювальну складність $O(n)$ і відповідає за **необоротність, відсутність колізій і цілісність даних**.

4. **MAC або HMAC (механізм автентичності повідомлень).** гешована MAC-функція (HMAC) використовується для перевірки цілісності й автентичності повідомлення. HMAC будується на основі геш-функції та секретного ключа і дозволяє виявити будь-яку зміну даних, навмисну чи випадкову. У гібридних схемах MAC, як правило, обчислюється від комбінації службових полів (зашифрований симетричний ключ, вектор ініціалізації, ciphertext), що унеможливлює непомітну модифікацію будь-якої частини криптопакета. MAC забезпечує **цілісність та автентичність**.

5. **Криптографічний генератор випадкових чисел (ГВВ/PRNG).** Випадкові числа використовуються для:

   - генерації симетричних ключів;
   - створення ініціалізаційних векторів (IV);
   - побудови паддінгу в асиметричних алгоритмах;
   - генерації сольових значень і параметрів у протоколах.

   Надійний ГВВ — критичний елемент безпеки гібридної схеми: недостатня ентропія або передбачуваність генератора веде до повної компрометації ключів і системи в цілому.

Таким чином, гібридна криптосистема поєднує:

- **швидкість симетричної криптографії**;

- **криптостійкість та зручність розподілу ключів в асиметричній криптографії**;
- **захист цілісності та автентичності через MAC/HMAC**;
- **криптографічну випадковість через PRNG**.

Завдяки цьому підходу гібридні криптосистеми набули широкого поширення — включно з TLS/SSL, PGP, SSH, VPN-протоколами та більшістю сучасних стандартів захищеної передачі інформації.

2.2 Огляд та порівняння запропонованих бібліотек

Ми дещо підкорегували завдання та замінили бібліотеку PyCrypto на PyCryptodome. PyCryptodome – це форк PyCrypto, який:

- активно підтримується, виходять релізи (в т.ч. 2024–2025)
- повністю зберігає простір імен Crypto.* (майже drop-in replacement),
- додає: AEAD-режими (GCM, CCM, EAX, OCB), прискорений AES (через AES-NI) та чимало криптографічних примітивів.

2.2.1 OpenSSL

OpenSSL є промисловим стандартом у середовищі UNIX/Linux і реалізує широкий спектр криптографічних примітивів: симетричні (AES, 3DES, ChaCha20), асиметричні (RSA, DSA, DH, ECC), геш-функції (SHA-1, SHA-2, SHA-3), генератори випадкових чисел, а також високорівневі протоколи TLS/SSL та X.509. Бібліотека написана мовою C і містить велику кількість низькорівневих оптимізацій, включно з використанням асемблерних вставок, апаратного прискорення (AES-NI, AVX2) та можливістю роботи в режимі FIPS.

Базові криптографічні операції, такі як AES та SHA-256, демонструють надзвичайно високий рівень продуктивності завдяки ручній оптимізації та глибокій інтеграції з апаратними можливостями сучасних процесорів. Інструмент `openssl speed` дозволяє провести бенчмарки практично для будь-якого алгоритму.

Попри високу продуктивність, OpenSSL характеризується досить низьким рівнем абстракції. Розробник має працювати з C-API, який передбачає ручне керування контекстами та буферами. Це ускладнює використання OpenSSL у навчальних і демонстраційних цілях, особливо коли основна увага приділяється алгоритмічним аспектам гібридної криптосистеми.

2.2.2 Crypto++

Crypto++—це кросплатформна бібліотека на C++, яка містить великий набір сучасних криптографічних алгоритмів: AES, Camellia, ChaCha20, RSA, Diffie–Hellman, ECDSA/ECDH, SHA-1/2/3, Blake2, а також режими автентифікованого шифрування (GCM, CCM). Архітектурно вона побудована у

вигляді "потоків" (*Filters, Sources, Sinks*), що забезпечує модульність та поєднання компонентів у складніші конструкції.

Оптимізація бібліотеки спрямована на ефективність, і Crypto++ демонструє досить високі показники продуктивності у задачах симетричного та асиметричного шифрування. Обчислення RSA та операції з еліптичними кривими виконуються значно швидше, ніж у неоптимізованих реалізаціях, завдяки продуманим алгоритмам роботи з великими числами.

2.2.3 Cryptlib

Cryptlib пропонує не лише набір криптографічних примітивів, але і повноцінний криптографічний стек, орієнтований на побудову захищених систем у промислових умовах. До його складу входять TLS/SSL, SSH, S/MIME, PGP/OpenPGP, механізми управління ключами, сертифікаційні центри (CA), OCSP, CMP, SCEP та інші компоненти.

На рівні примітивів бібліотека надає AES, Blowfish, ChaCha20, DES/3DES, RC2/RC4, SHA-1/2, HMAC, алгоритми відкритого ключа (RSA, DSA, ECDH/ECDSA). Крім того, архітектура Cryptlib дозволяє гнучко конфігурувати склад бібліотеки для обмежених середовищ, що робить її привабливою для вбудованих систем.

2.2.4 PyCryptodome

PyCryptodome є сучасною, підтримуваною та розширеною альтернативою застарілій бібліотеці PyCrypto. Вона зберігає простий і зрозумілий Python-інтерфейс, але реалізує криптографічні операції у вигляді високоефективних C-розширень, що забезпечує прийнятну продуктивність. Бібліотека містить модулі для симетричного шифрування (AES, DES, ChaCha20), гешування (MD5, SHA-1, SHA-256, SHA-3, Blake2), алгоритмів відкритого ключа (RSA, DSA, ElGamal), генераторів випадкових чисел, а також сучасні AEAD-режими (GCM, EAX, OCB).

Для нашого проєкту ми зупинились на бібліотеці PyCryptodome, оскільки вона була найлегшою в реалізації. Низький рівень абстракції Python, попри збереження функціональності та обчислювальної ефективності, робить її найзручнішою саме для цього кейсу:

- Набір примітивів повністю достатній для гібридної схеми: AES, DES, ARC4, RSA, ElGamal, DSA, SHA-1/256, HMAC, KDF, PRNG;
- Для гібридної схеми більшість часу витрачається на симетричне шифрування та хешування; в PyCrypto ці частини реалізовані на C і досить швидкі для лабораторних експериментів на Linux.

В таблиці [**?**] можна ознайомитись з більш наглядним порівнянням функціональності всіх наведених бібліотек.

2.3 Порівняння ефективності бібліотек за часом виконання

У цьому підрозділі розглянемо експериментальні дані щодо продуктивності бібліотек OpenSSL, Crypto++, Cryptlib та PyCryptodome під Linux. Нижче подано інтерпретацію результатів саме з точки зору побудови гібридної криптосистеми (інтенсивне симетричне шифрування + хешування, рідкісні операції з відкритим ключем).

2.3.1 OpenSSL

OpenSSL є де-факто стандартом для криптографії під Linux і, згідно з доступними вимірюваннями, **демонструє найвищу швидкодію серед розглянутих бібліотек**. Це підтверджується як практикою використання у високонавантажених сервісах, так і спеціальними тестами.

У роботі Kannojia & Kurmi (2022) виконано порівняльний аналіз шести TLS-бібліотек (OpenSSL, GnuTLS, BoringSSL, AWS s2n, NSS, Cryptlib) на п'яти операційних системах, з яких три — Linux-дистрибутиви: *Ubuntu*, *Fedora*, *Debian-etch*. Для кожної ОС вимірювалися середні значення throughput (KB/s) для:

- асиметричних операцій (Sign/s, Verify/s);
- хеш-функцій (HMAC-MD5/SHA1/SHA256/...);
- симетричних шифрів (AES-GCM, AES-CBC, 3DES тощо).

За результатами таблиці 3 статті середні значення для операцій підпису/верифікації (усереднені по трьох Linux-дистрибутивах) для OpenSSL становлять приблизно 1350 Sign/s та $2.68 \cdot 10^4$ Verify/s, тобто бібліотека забезпечує тисячі операцій відкритого ключа за секунду для ключів 1024–15360 біт. При цьому Cryptlib за тими ж тестами показує дуже близькі величини для RSA та інших KEM-шифрів, що означає порівнянну швидкість саме на рівні асиметрії.

Найбільш важливими для гібридної криптосистеми є симетричні алгоритми. Таблиця 9 статті демонструє, що середній throughput симетричних шифрів OpenSSL по трьох Linux-дистрибутивах становить приблизно 33,260 KB/s, тоді як Cryptlib показує лише 9,590 KB/s, тобто OpenSSL більш ніж утричі швидший за Cryptlib при роботі з AES/3DES та іншими блоковими шифрами на тих самих тестах. Аналогічно, для хеш-функцій (табл. 6) середній throughput OpenSSL складає близько 15,711 KB/s проти 14,376 KB/s у Cryptlib, що дає помітну, але вже не таку драматичну перевагу.

Додаткові *micro-benchmark-и* (утиліта `openssl speed`) показують, що на сучасних x86_64-процесорах з підтримкою AES-NI OpenSSL досягає сотень мегабайт на секунду для AES-128-CBC та AES-256-CBC, а також дуже низької величини cycles/byte для SHA-256. Це підтверджує, що OpenSSL вміє максимально використовувати апаратне прискорення (AES-NI, AVX2,

AVX-512) і залишається найшвидшою бібліотекою з тих, що порівнюються, зокрема на Linux-серверах.

2.3.2 Crypto++

Crypto++ орієнтована на C++ і містить вбудовану систему бенчмаркінгу, яка запускається через тестову програму `cryptest` з підкомандою `b` ("benchmarks"). На Linux це дозволяє автоматично прогнати усі підтримувані алгоритми та отримати значення в MiB/s і cycles/byte.

Офіційна вікі-сторінка Crypto++ з прикладами бенчмарків на процесорі Intel Core i5 Skylake (2.7 ГГц) демонструє для AES-128 у режимі CTR швидкодію близько 0.58 cycles/byte, що відповідає приблизно 4.4 GiB/s, а для режиму CCM — порядку 3.0 cycles/byte і $\approx$ 0.86 GiB/s. Ці показники знаходяться на одному порядку з результатами OpenSSL, отриманими утилітою `openssl speed` для AES-128-CBC на трошки гіршому обладнанні.

Таким чином, при роботі *без* накладних витрат інтерпретатора (чистий C++) Crypto++ забезпечує продуктивність, близьку до OpenSSL для симетричних шифрів і хешів. Різниця між двома бібліотеками частіше визначається конкретними оптимізаціями під конкретну архітектуру й наявністю/відсутністю асемблерних вставок, ніж принциповою різницею у виборі алгоритмів.

Для гібридної криптосистеми під Linux це означає, що *з погляду часу* Crypto++ є дуже швидким варіантом, який поступається OpenSSL здебільшого лише через менш агресивну оптимізацію під конкретні CPU та відсутність настільки ж глибоко відполірованого асемблерного коду.

2.3.3 Cryptlib

Для Cryptlib у згаданій роботі Kannojia & Kurmi (2022) наведено ті самі типи тестів, що й для OpenSSL: throughput операцій Sign/Verify для набору KEM-шифрів, throughput для хеш-функцій та симетричних шифрів. Узагальнення по трьох Linux-дистрибутивах (Ubuntu, Fedora, Debian-etch) показує:

- для симетричних шифрів середній throughput Cryptlib становить близько 9,590 KB/s проти 33,260 KB/s у OpenSSL;
- для хеш-функцій Cryptlib демонструє $\approx$ 14,376 KB/s проти $\approx$ 15,711 KB/s у OpenSSL;
- для асиметричних операцій (Sign/s, Verify/s) Cryptlib має значення, дуже близькі до OpenSSL, іноді навіть трохи вищі для Sign/s, але ця різниця не впливає на загальний час гібридної схеми.

У власній документації Cryptlib зазначається, що на сучасних системах більшість блокових шифрів і хеш-функцій дають "десятки–сотні мегабайт на секунду а операції відкритого ключа виконуються за мілісекунди, але ці

цифри узгоджуються з тим, що на тих самих тестах OpenSSL систематично показує вищий throughput.

Отже, у порівнянні з OpenSSL, Cryptlib можна охарактеризувати як *помітно повільніший* для масового симетричного шифрування і лише трохи слабший для хешування, при приблизно порівняній швидкості операцій з відкритим ключем. Для побудови гібридної криптосистеми, де домінує саме симетрична частина, це означає, що Cryptlib програє OpenSSL за часом виконання, навіть якщо пропонує більш "інтегрований" стек протоколів і сервісів.

2.3.4 PyCryptodome (Python)

PyCryptodome є наступником PyCrypto з практично сумісним API, а отже, для оцінки часової ефективності можна орієнтуватися на результати бенчмаркінгу PyCrypto. У репозиторії `pyCryptoBenchmarking` наведено порівняння PyCrypto та бібліотеки `cryptography` (обидві поверх OpenSSL/low-level C) для симетричних шифрів, хеш-функцій, RSA та генерації випадкових чисел.

Для симетричних шифрів (AES, 3DES, Blowfish тощо) у вимірюванні "ітерацій за секунду" реалізація PyCrypto демонструє вищу швидкість, ніж `cryptography`, причому для більшості алгоритмів у 1.5–2 рази на типових розмірах буфера. Для хеш-функцій ситуація цікавіша: при великих повідомленнях (64 KB) `cryptography` випереджає PyCrypto за SHA-256/384/512, але для малих повідомлень (32 байти) PyCrypto є значно швидшим завдяки меншому накладному інтерфейсному overhead-у. Для RSA бібліотека `cryptography` працює швидше, тоді як PyCrypto дає результати, близькі до "чистої" `pow()` над великими числами.

Оскільки критичні частини PyCryptodome також написані на C, а API і структура дуже близькі до PyCrypto, можна вважати, що його часова ефективність буде тією ж або кращою в межах Python-оточення. Водночас потрібно враховувати, що в усіх випадках до швидкодії додається накладна вартість інтерпретатора Python: навіть якщо внутрішня C-реалізація AES чи SHA-256 порівнянна з OpenSSL чи Crypto++, повна швидкодія ("end-to-end") буде помітно нижчою через часті переходи між Python та C, алокацію об'єктів, GC тощо.

У підсумку PyCryptodome можна охарактеризувати так:

- для невеликих блоків даних (типові повідомлення, ключі, службові структури) швидкодії достатньо, щоб *зручний Python-код* був прийнятним навіть для багаторазового шифрування/дешифрування;
- у порівнянні з "чистим" C-рівнем OpenSSL або Crypto++ PyCryptodome суттєво повільніший, особливо при обробці великих потоків даних, але це плата за високу наочність і простоту реалізації гібридної схеми в лабораторній роботі.

2.3.5 Загальні висновки для гібридної криптосистеми

Якщо розглядати саме гібридну криптосистему під Linux, де основним вузьким місцем є симетричне шифрування великих обсягів даних та хешування, можна підсумувати:

- **OpenSSL** — найшвидша з розглянутих бібліотек, особливо для симетричних алгоритмів на Linux з підтримкою AES-NI; є природним вибором для високонавантажених серверів.
- **Crypto++** забезпечує продуктивність того ж порядку, що й OpenSSL, і є хорошим варіантом для C++-проектів, де важлива як швидкодія, так і гнучкий об'єктно-орієнтований API.
- **Cryptlib** має прийнятну, але нижчу за OpenSSL швидкодію, особливо на рівні симетричних шифрів; його переваги більше в "повному стеку" протоколів, а не в чистому throughput.
- **PyCryptodome** помітно повільніший за нативні C/C++ бібліотеки через накладні витрати Python, але абсолютно достатній для навчальної гібридної криптосистеми, де важливіше прозорість реалізації та легкість експериментів, ніж максимальна швидкість.

2.4 Реалізація гібридної криптосистеми на основі PyCryptodome

Розглянемо конкретну програмну реалізацію гібридної криптосистеми під Linux, яка використовує бібліотеку PyCryptodome (модулі `Crypto.PublicKey`, `Crypto.Cipher`, `Crypto.Random`) у поєднанні зі стандартною бібліотекою Python (`hashlib`, `hmac`, `struct`). Схема поєднує симетричне шифрування AES-256 у режимі CBC, асиметричне шифрування RSA-2048 з паддінгом OAEP на базі SHA-256 та механізм контролю цілісності HMAC-SHA256.

Загальна структура. Гібридна схема складається з таких основних компонентів:

- генерація пари ключів RSA розміром 2048 біт;
- одноразова генерація випадкового сеансового ключа AES-256 та вектора ініціалізації (IV) для режиму CBC;
- шифрування відкритого тексту алгоритмом AES-256-CBC з PKCS#7-паддінгом;
- шифрування сеансового ключа AES за допомогою RSA-OAEP(SHA-256);
- обчислення HMAC-SHA256 від конкатенації службових полів (`enc_key` ‖ `iv` ‖ `ciphertext`) з окремим секретним ключем HMAC;

- упаковка всіх компонентів у єдиний байтовий контейнер (*blob*) фіксованої структури.

Для використання SHA-256 у режимі OAEP створюється мінімальний "адаптер" `HashlibSHA256`, який надає PKCS1_OAEP інтерфейс, сумісний із очікуваним хеш-модулем (атрибути `digest_size`, `block_size`, `oid` та статичний метод `new()` на базі `hashlib.sha256`).

Генерація ключів RSA. Функція

`generate_rsa_keys(bits=2048)`

використовує `RSA.generate(bits)` для побудови пари ключів розміром 2048 біт. Приватний та публічний ключ експортуються у форматі PEM за допомогою методів `export_key('PEM')`. Повертається пара (`priv_pem`, `pub_pem`), яка надалі використовується відповідно для розшифрування та шифрування.

Алгоритм гібридного шифрування. Функція

`hybrid_encrypt(plaintext, pub_pem)` $\rightarrow$ `blob`

реалізує шифрування повідомлення у вигляді наступної послідовності дій:

1. Імпортується публічний ключ `pub_pem` за допомогою `RSA.import_key()`, конструюється шифр `PKCS1_OAEP.new(rsa_key, hashAlgo=HashlibSHA256)`, який реалізує RSA-OAEP із використанням SHA-256.

2. Генерується випадковий симетричний ключ AES-256: `aes_key` = `get_random_bytes(32)`, та 128-бітний вектор ініціалізації `iv` = `get_random_bytes(16)`. Джерелом випадковості є криптографічно стійкий ГВВ Linux (`/dev/urandom` через PyCryptodome).

3. Створюється об'єкт `AES.new(aes_key, AES.MODE_CBC, iv=iv)` та виконується PKCS#7-паддінг: до відкритого тексту додаються k байтів із значенням k, де $k = 16 - (|\texttt{plaintext}| \bmod 16)$. Після цього виконується шифрування: `ciphertext` = `aes_cipher.encrypt(padded)`.

4. Сеансовий ключ AES шифрується RSA-OAEP: `enc_key` = `rsa_cipher.encrypt(aes_key)`. Отриманий `enc_key` має довжину, що відповідає розміру модуля RSA (для 2048 біт — 256 байт).

5. Генерується окремий 256-бітний ключ для HMAC: `hmac_key` = `get_random_bytes(32)`. Обчислюється тег HMAC-SHA256 над конкатенацією `enc_key` $\|$ `iv` $\|$ `ciphertext`:

   `tag` = `hmac.new(hmac_key, enc_key + iv + ciphertext, hashlib.sha256).digest()`.

6. Сформований криптографічний контейнер (*blob*) має структуру:

$$\underbrace{\texttt{len(enc\_key)}}_{\text{2 байти (big-endian)}} \| \texttt{enc\_key} \| \texttt{hmac\_key} \| \texttt{iv} \| \texttt{ciphertext} \| \texttt{tag},$$

де довжина `enc_key` кодується у форматі big-endian (`»H"`) за допомогою модуля `struct`. Цей байтовий рядок і повертається як результат.

У підсумку гібридне шифрування забезпечує конфіденційність даних (AES-256-CBC), конфіденційність сеансового ключа (RSA-OAEP(SHA-256)) та цілісність/автентичність контейнера (HMAC-SHA256).

Алгоритм гібридного розшифрування. Функція

`hybrid_decrypt(blob, priv_pem)` $\rightarrow$ `plaintext`

реалізує операцію, обернену до `hybrid_encrypt`.

Спочатку `blob` розпаковується у відповідності до визначеного формату. Зчитується двобайтне поле довжини `enc_key_len`, після чого вилучаються: `enc_key` (зашифрований AES-ключ), `hmac_key` (32 байти), `iv` (16 байтів), `ciphertext` (усі байти, окрім останніх 32), та `tag` (останні 32 байти — HMAC-SHA256).

Далі обчислюється "очікуваний" тег:

`calc_tag` = `hmac.new(hmac_key, enc_key + iv + ciphertext, hashlib.sha256).digest()`,

і виконується порівняння з отриманим `tag` за допомогою `hmac.compare_digest()`, що захищає від таймінг-атак. При невідповідності тегів кидається виключення `ValueError` (*"HMAC verification failed"*), а розшифрування припиняється. Це гарантує, що будь-яка модифікація контейнера буде виявлена до спроби розшифрувати AES-ключ або дані.

Якщо HMAC успішно перевірено, імпортується приватний ключ `priv_pem` через `RSA.import_key()`, створюється об'єкт `PKCS1_OAEP.new(rsa_key, hashAlgo=HashlibSHA256)` і виконується розшифрування `enc_key` для отримання оригінального `aes_key`. Потім конструюється `AES.new(aes_key, AES.MODE_CBC, iv=iv)` і виконується розшифрування `ciphertext` у `padded`.

Заключний крок — видалення PKCS#7-паддінгу. Останній байт `padded[-1]` інтерпретується як кількість байтів паддінгу `pad_len`. Якщо `pad_len` виходить за межі $[1, 16]$, кидається `ValueError` (*"Невірний padding"*). Інакше останні `pad_len` байтів відкидаються, і отриманий `plaintext` повертається як результат.

Контрольний приклад. У блоці `if __name__ == "__main__":` виконується тестовий запуск: генерується пара ключів RSA-2048, формується демонстраційне повідомлення (кілька рядків ASCII-тексту), викликаються `hybrid_encrypt` та `hybrid_decrypt`, після чого виводиться початковий та відновлений текст,

а також логічне значення порівняння `message == recovered`. У разі коректної роботи системи на екрані буде виведено `OK: True`, що підтверджує правильність реалізації гібридної криптосхеми під Linux із використанням PyCryptodome та стандартних засобів Python.

Висновки

У ході виконання лабораторної роботи ми дослідили сучасні криптографічні бібліотеки, що використовуються для побудови гібридних криптосистем у середовищі Linux, а також здійснили порівняльний аналіз їхньої ефективності за часом виконання та структурними властивостями.

Ми встановили, що серед розглянутих рішень **OpenSSL** є найпродуктивнішою бібліотекою, що підтверджується результатами експериментального тестування на трьох Linux-дистрибутивах (Ubuntu, Fedora, Debian). Висока пропускна здатність симетричних алгоритмів (AES) та хеш-функцій, а також ефективне використання апаратного прискорення (AES-NI, AVX2) робить OpenSSL оптимальним вибором для високонавантажених систем.

Бібліотека **Crypto++** демонструє продуктивність, близьку до OpenSSL у середовищі C++, що підтверджується вбудованими бенчмарками та незалежними вимірюваннями. Вона є потужним інструментом для побудови високошвидкісних криптосистем на C++.

Cryptlib, хоча й забезпечує широкий набір протоколів та інфраструктурних можливостей, поступається OpenSSL у швидкодії, особливо на рівні симетричного шифрування. Проте вона залишається універсальним компонентом для створення комплексних безпекових рішень.

У свою чергу, **PyCryptodome** хоч і не може конкурувати з нативними C/C++ бібліотеками за продуктивністю, проте забезпечує прийнятний рівень ефективності для лабораторних та дослідницьких завдань. Проста та наочна структура API дозволила нам реалізувати повноцінну гібридну криптосистему, яка поєднує AES-256-CBC, RSA-OAEP(SHA-256) та HMAC-SHA256, зберігаючи чіткість логіки роботи та відтворюваність результатів.

Здійснена нами реалізація підтверджує теоретичні властивості гібридних схем: симетричне шифрування забезпечує високу швидкість, асиметричне — безпечний розподіл ключів, а механізми HMAC гарантують цілісність та автентичність даних. Linux, завдяки наявності системних джерел ентропії, апаратній підтримці криптографії та стабільним механізмам роботи з пам'яттю, є оптимальною платформою для побудови таких систем.

У підсумку, ми не лише проаналізували теоретичні особливості криптографічних бібліотек, а й розробили практичну гібридну криптосистему, що підтверджує ефективність обраного підходу та демонструє взаємодію криптографічних примітивів у реальному середовищі Linux.

Табл. 1: Порівняння криптографічних бібліотек для побудови гібридної криптосистеми

Критерій	**OpenSSL**	**Crypto++**	**Cryptlib**	**PyCryptodome**
Мова реалізації	C	C++	C	Python (з використанням C-розширень)
Підтримка AES, RSA, SHA-256	Так; дуже широка підтримка, включно з апаратними оптимізаціями	Так; повний набір сучасних примітивів	Так; включає також TLS/SSH/PKI протоколи	Так; повна підтримка основних примітивів
Підтримка AEAD-режимів (GCM, EAX, CCM)	Так	Так	Так (через високорівневі інтерфейси)	Так
Оптимізація під Linux	Висока; використання асемблерних оптимізацій та AES-NI	Висока; оптимізований C++ код	Хороша; орієнтована на портативність	Середня; залежить від Python overhead, але критичні частини оптимізовано в C
Рівень API	Низький/середній (Evp-API, потребує детального налаштування)	Середній; модульна “pipeline”-архітектура	Високий; орієнтована на комплексні системи безпеки	Середній; простий та інтуїтивний Python-API
Розмір та складність бібліотеки	Велика кодова база, багато протоколів і форматів	Середня; орієнтована на розробників C++	Середня/велика; включає повний криптографічний стек	Мала; зручна для швидкої розробки та експериментів
Наочність та зручність використання у навчальних цілях	Середня; низькорівневий API	Середня; досить складний шаблонний код	Низька; орієнтована на комплексні системи	Висока; ідеальна для демонстрації гібридних криптосистем у навчальних роботах
Актуальний стан розвитку	Активно підтримується; промисловий стандарт	Активно підтримується	Активно підтримується	Активно підтримується; сучасний замінник PyCrypto