НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. Ігоря

СІКОРСЬКОГО»

НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ

ЗВІТ З ЛАБОРАТОРНОЇ РОБОТИ №1

З дисципліни «Методи реалізації криптографічних механізмів»

«ВИБІР ТА РЕАЛІЗАЦІЯ БАЗОВИХ ФРЕЙМВОРКІВ ТА БІБЛІОТЕК»

Виконав:

студент групи ФБ-41мн

Білик Д. М.

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 (cryptlib), PyCrypto/PyCryptodome для розробки гібридної криптосистеми під Linux, з урахуванням:
1) підтримуваних криптографічних примітивів (симетричних, асиметричних, гешів, MAC, KDF);
2) ефективності за часом та пам’яттю на платформі Linux;
3) функціональних можливостей 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. Симетричні криптосистеми забезпечують високу швидкість шифрування та лінійну часову складність O(n) відносно розміру даних, але потребують спільного секретного ключа в обох сторін. Натомість асиметричні криптосистеми дають можливість безпечного обміну ключами без попередньої домовленості, проте мають значно більші обчислювальні накладні витрати. Отже, гібридна схема використовує асиметрію лише для захисту короткого сеансового ключа, а основний обсяг даних шифрується швидким симетричним алгоритмом.

Типова гібридна криптосистема складається з таких примітивів:

Симетричний алгоритм шифрування (AES-128/192/256).
Асиметричне шифрування або обмін ключами (RSA-OAEP, ECDH/ECIES).
Криптографічна геш-функція (SHA-256/384/512) для дайджестів та протокольних конструкцій.
MAC/HMAC для забезпечення цілісності та автентичності криптопакета (якщо не використовується AEAD).
Криптографічно стійкий генератор випадкових чисел (PRNG/CSPRNG) для ключів, IV/nonce, сольових значень.

2.2 Огляд та порівняння запропонованих бібліотек

PyCrypto є застарілою та не підтримується, для практичної реалізації на Python я використав PyCryptodome — сумісний форк (практично drop-in replacement) з простором імен Crypto.*, який додає сучасні режими (GCM/CCM/EAX/OCB) та оптимізовані реалізації на C.

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-1/2/3), генератори випадкових чисел, а також високорівневі протоколи TLS/SSL та X.509. Бібліотека написана мовою C і містить низькорівневі оптимізації, включно з використанням апаратного прискорення (AES-NI, AVX тощо). Для бенчмаркінгу на Linux часто використовують утиліту 'openssl speed'.

2.2.2 Crypto++

Crypto++ — кросплатформна бібліотека на C++, яка містить великий набір сучасних криптографічних алгоритмів: AES, ChaCha20, RSA, DH, ECDSA/ECDH, SHA-1/2/3, Blake2 тощо, а також режими автентифікованого шифрування (GCM, CCM). Архітектурно бібліотека побудована у вигляді «потоків» (Sources/Filters/Sinks), що дозволяє модульно комбінувати компоненти. Для тестування продуктивності використовується утиліта 'cryptest'.

2.2.3 CryptoLib

CryptoLib — позиціонується як комплексний криптографічний стек, орієнтований на побудову захищених систем: окрім примітивів, може включати механізми керування ключами та підтримку протоколів/форматів (TLS/SSL, SSH, S/MIME, OpenPGP тощо). На рівні примітивів бібліотека зазвичай надає AES, ChaCha20, DES/3DES, SHA-1/2, HMAC, а також RSA/DSA/ECDH/ECDSA. Її перевага — більш високорівневий підхід і “інтегрованість”, недолік — менша поширеність порівняно з OpenSSL.

2.2.4 PyCrypto / PyCryptodome (Python)

PyCrypto — історична Python-бібліотека для реалізації криптографічних примітивів. PyCryptodome зберігає простий Python-інтерфейс і використовує C-розширення для прискорення критичних операцій, що забезпечує прийнятну продуктивність для лабораторних експериментів.

2.2.5 Порівняння функціональних можливостей

Таблиця 1 – Порівняння криптографічних бібліотек для побудови гібридної криптосистеми (Linux)

Критерій	OpenSSL	Crypto++	Cryptlib	PyCryptodome
Мова реалізації	C	C++	C	Python (C-розширення)
Підтримка AES/RSA/SHA-256	Так	Так	Так	Так
Підтримка AEAD (GCM/CCM/EAX)	Так	Так	Так (через API)	Так

Рівень API	Низький/середній (EVP)	Середній (потоки)	Високий (toolkit)	Середній (Python API)
Зручність для навчальних цілей	Середня	Середня	Низька/середня	Висока
Типовий сценарій	Продакшн/TLS/PKI	C++ застосунки	Комплексні security-системи	Швидкі прототипи/лаби

2.3 Порівняння ефективності бібліотек за часом та пам’яттю (методика)

Для оцінки ефективності під Linux доцільно порівнювати продуктивність симетричних алгоритмів (AES), хешування (SHA-256) та операцій відкритого ключа (RSA) окремо, оскільки у гібридній схемі домінує час симетричного шифрування та обчислення MAC/AEAD.

Практична методика вимірювання:
• OpenSSL: використати 'openssl speed -evp aes-256-gcm sha256 rsa2048'.
• Crypto++: запустити 'cryptest b' та вибрати ключові алгоритми.
• Cryptlib: використати вбудовані тестові утиліти/приклади або профілювання викликів API.
• PyCryptodome: виконати Python-скрипт з time.perf_counter() на буферах різного розміру (наприклад 4KB, 64KB, 1MB) і порівняти середній час шифрування/хешування.

2.4 Реалізація гібридної криптосистеми на Python (Linux) на основі PyCryptodome

Далі наведено реалізацію гібридної криптосистеми, яка поєднує:
• симетричне шифрування AES-256-CBC з PKCS#7-паддінгом, який передбачає доповнення повідомлення k байтами зі значенням k, де k — кількість доданих байтів;
• асиметричне шифрування сеансового ключа RSA-2048 з OAEP на базі SHA-256;
• контроль цілісності та автентичності контейнера за допомогою HMAC-SHA256.

Контейнер (blob) має структуру:
len(enc_key) (2 байти, big-endian) ‖ enc_key ‖ hmac_key ‖ iv ‖ ciphertext ‖ tag
де tag = HMAC-SHA256(enc_key ‖ iv ‖ ciphertext).

2.4.1 Перелік функцій та опис інтерфейсів

Функції реалізації:
generate_rsa_keys(bits=2048) → (priv_pem, pub_pem)
hybrid_encrypt(plaintext: bytes, pub_pem: bytes) → blob: bytes
hybrid_decrypt(blob: bytes, priv_pem: bytes) → plaintext: bytes

Вхідні дані: plaintext (bytes), RSA ключі у PEM (bytes).
Вихідні дані: криптоконтейнер blob (bytes) або відновлений plaintext.
Коди повернення: у Python для помилок використовується механізм виключень (ValueError/TypeError), для успіху — повертається значення функції.

Код реалізації

```
from Crypto.PublicKey import RSA
from Crypto.Cipher import AES, PKCS1_OAEP
from Crypto.Random import get_random_bytes
from Crypto.Hash import SHA256, HMAC
import struct


def pkcs7_pad(data: bytes, block_size: int = 16) -> bytes:
    pad_len = block_size - (len(data) % block_size)
    return data + bytes([pad_len] * pad_len)


def pkcs7_unpad(data: bytes, block_size: int = 16) -> bytes:
    if not data or len(data) % block_size != 0:
        raise ValueError("Wrong block size for PKCS#7 unpad")
    pad_len = data[-1]
    if pad_len < 1 or pad_len > block_size:
        raise ValueError("wrong padding")
    if data[-pad_len:] != bytes([pad_len] * pad_len):
        raise ValueError("wrong padding")
```

```
    return data[:-pad_len]


def generate_rsa_keys(bits: int = 2048):
    key = RSA.generate(bits)
    priv_pem = key.export_key(format='PEM')
    pub_pem = key.publickey().export_key(format='PEM')
    return priv_pem, pub_pem


def hybrid_encrypt(plaintext: bytes, pub_pem: bytes) -> bytes:
    # 1) RSA-OAEP(SHA-256)
    pub_key = RSA.import_key(pub_pem)
    rsa_cipher = PKCS1_OAEP.new(pub_key, hashAlgo=SHA256)
    aes_key = get_random_bytes(32)
    iv = get_random_bytes(16)
    aes = AES.new(aes_key, AES.MODE_CBC, iv=iv)
    padded = pkcs7_pad(plaintext, 16)
    ciphertext = aes.encrypt(padded)


    enc_key = rsa_cipher.encrypt(aes_key)


    hmac_key = get_random_bytes(32)
    h = HMAC.new(hmac_key, enc_key + iv + ciphertext, digestmod=SHA256)
    tag = h.digest()


    blob = struct.pack(">H", len(enc_key)) + enc_key + hmac_key + iv +
ciphertext + tag
    return blob
```

```python
def hybrid_decrypt(blob: bytes, priv_pem: bytes) -> bytes:
    if len(blob) < 2:
        raise ValueError("Wrong continer format")
    enc_len = struct.unpack(">H", blob[:2])[0]
    off = 2
    enc_key = blob[off:off+enc_len]; off += enc_len
    hmac_key = blob[off:off+32]; off += 32
    iv = blob[off:off+16]; off += 16

    tag_size = SHA256.digest_size
    ciphertext = blob[off:-tag_size]
    tag = blob[-tag_size:]

    h = HMAC.new(hmac_key, enc_key + iv + ciphertext, digestmod=SHA256)
    try:
        h.verify(tag)
    except ValueError:
        raise ValueError("HMAC verification failed: Data was changed!")

    priv_key = RSA.import_key(priv_pem)
    rsa_cipher = PKCS1_OAEP.new(priv_key, hashAlgo=SHA256)
    aes_key = rsa_cipher.decrypt(enc_key)
    aes = AES.new(aes_key, AES.MODE_CBC, iv=iv)
    padded = aes.decrypt(ciphertext)
    plaintext = pkcs7_unpad(padded, 16)
    return plaintext
```

```python
if __name__ == "__main__":
    print("Key generation...")
    priv_pem, pub_pem = generate_rsa_keys(2048)
    message = (
        b"Lab1 message for encoding.\n"
        b"AES-256-CBC + RSA-OAEP(SHA-256) + HMAC-SHA256."
    )
    print("Cyphering...")
    blob = hybrid_encrypt(message, pub_pem)

    print("Decyphering...")
    recovered = hybrid_decrypt(blob, priv_pem)

    print("-" * 20)
    print("Result:", recovered.decode())
    print("Integrity check OK:", recovered == message)
```

```
(.venv) friedreich@friedreich:~/Documents/Labs$ /home/friedreich/Documents/Labs/.venv/bin/python /home/friedreich/Documents/Labs/MPKM/Lab1/Lab1.py
Key generation...
Cyphering...
Decyphering...
--------------------
Result: Lab1 message for encoding.
AES-256-CBC + RSA-OAEP(SHA-256) + HMAC-SHA256.
Integrity check OK: True
```

Висновки

У ході виконання лабораторної роботи було розглянуто бібліотеки OpenSSL, Crypto++, Cryptlib та PyCrypto/PyCryptodome для побудови гібридної криптосистеми під Linux. OpenSSL є найпоширенішим варіантом для продакшн-середовищ завдяки високій продуктивності та підтримці протоколів TLS/PKI. Crypto++ є потужною альтернативою для C++ застосунків, що забезпечує високу швидкодію і гнучку архітектуру. Cryptlib орієнтована на високорівневі security-сервіси і може бути зручною у комплексних системах. Для лабораторної реалізації на Python найзручнішим вибором є PyCryptodome, оскільки він забезпечує прозорий API та достатню продуктивність, дозволяючи реалізувати повну гібридну схему.