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»

Виконала:
студентка групи ФІ-52мн
Балацька В. В.

КИЇВ – 2025

Мета роботи: Вибір базових бібліотек/сервісів для подальшої реалізації криптосистеми
Завдання: розробка технічних вимог (із вибором або бібліотеки реалізації арифметичних операцій або бібліотеки реалізації основних криптографічних примітивів) для різних варіантів реалізацій ІТ-систем. Підгрупа 3А. Вибір бібліотеки для реалізації Web-сервісу електронного цифрового підпису

Хід роботи

Для виконання даної лабораторної роботи було розроблено наступний план:

1. **Аналіз предметної області**
    a. Ознайомитись із принципами роботи електронного цифрового підпису.
    b. Визначити вимоги до ІТ-систем, які потребують використання ЕЦП (напр. електронні документи, сервіси онлайн-автентифікації, банківські системи).
    c. Виділити ключові функції, які має забезпечувати бібліотека для ЕЦП (генерація ключів, підписування, перевірка підпису, підтримка алгоритмів).
2. **Вибір варіанту бібліотеки**
    a. Провести огляд існуючих бібліотек для реалізації арифметичних операцій та криптографічних примітивів.
    b. Визначити критерії вибору бібліотеки:
        i. безпека та стандарти (наприклад, відповідність PKCS#7, PKCS#11, ГОСТ, RSA, ECDSA);
        ii. продуктивність;
        iii. зручність інтеграції з Web-сервісом;
        iv. ліцензія (open-source чи комерційна).
    c. Обрати конкретну бібліотеку для реалізації Web-сервісу ЕЦП (наприклад, OpenSSL, BouncyCastle, libsodium, PyCryptodome).
3. **Формування технічних вимог**
    a. Скласти перелік функціональних вимог (що повинна вміти система):
        i. створення пари ключів;
        ii. підписування даних;
        iii. перевірка підпису;
        iv. підтримка сертифікатів;
        v. логування операцій.
    b. Скласти перелік нефункціональних вимог:
        i. `продуктивність (час підпису ≤ X мс);`
        ii. масштабованість (робота з N користувачами одночасно);
        iii. безпека (захист приватного ключа, стійкість до атак).

4. **Проектування Web-сервісу**
    a. Визначити архітектуру Web-сервісу (REST API чи SOAP).
    b. Скласти схему взаємодії клієнта та сервера.
    c. Визначити основні ендпоїнти (наприклад: /sign, /verify, /keys/generate).
5. **Обґрунтування вибору**
    a. Пояснити, чому саме ця бібліотека відповідає завданню.
    b. Провести порівняння з альтернативними варіантами.
6. **Висновки**
    a. Підсумувати, як сформульовані технічні вимоги відповідають задачам Web-сервісу ЕЦП.
    b. Дати рекомендації для практичної реалізації та можливих подальших доопрацювань.

1) Аналіз предметної області

1.1 Принципи роботи електронного цифрового підпису

Електронний цифровий підпис (ЕЦП) є криптографічним механізмом, який забезпечує автентичність та цілісність електронних даних. Його основні принципи роботи базуються на використанні асиметричної криптографії:

- Ключова пара: кожен користувач має приватний ключ (для підпису) та публічний ключ (для перевірки).
- Процес підпису: дані хешуються криптографічною хеш-функцією (SHA-256, SHA-3 тощо), після чого отриманий хеш шифрується приватним ключем. Результат шифрування і є цифровим підписом.
- Перевірка підпису: одержувач за допомогою публічного ключа розшифровує підпис і порівнює його з обчисленим самостійно хешем даних. Якщо вони збігаються — підпис дійсний.

Таким чином, ЕЦП гарантує:

- автентифікацію (підтвердження особи підписанта);
- цілісність даних (захист від змін після підпису);
- невідмовність (автор не може заперечити факт підпису).

1.2 Вимоги до IT-систем, що потребують використання ЕЦП

ЕЦП застосовується в багатьох сферах, де важлива юридична значимість електронних документів і безпека обміну даними. Основні вимоги до таких систем:

1. Системи електронного документообігу
    - підтвердження легітимності електронних договорів, актів, накладних;
    - можливість багаторазової перевірки підпису.
2. Онлайн-сервіси автентифікації
    - забезпечення безпечного входу користувачів без паролів;

    - інтеграція з державними системами ідентифікації (наприклад, BankID, ID.GOV.UA).
3. Банківські та фінансові системи
    - підтвердження транзакцій клієнтів;
    - захист від шахрайства при віддалених операціях.
4. Електронна комерція та державні послуги (e-Gov, e-Commerce)
    - підтвердження замовлень, заявок та звернень;
    - використання в системах публічних закупівель.

1.3 Ключові функції бібліотеки для ЕЦП

Бібліотека, яка використовується для реалізації електронного цифрового підпису у Web-сервісі, повинна забезпечувати такі функції:

- Генерація криптографічних ключів: створення надійних пар ключів (RSA, ECDSA, Ed25519 тощо).
- Підписування даних: формування цифрового підпису з використанням приватного ключа.
- Перевірка підпису: перевірка коректності та достовірності підпису за допомогою публічного ключа.
- Підтримка алгоритмів і стандартів: робота з різними криптографічними алгоритмами (RSA, ECDSA, ГОСТ, SHA-2, SHA-3), а також відповідність міжнародним стандартам (PKCS#7, X.509, CMS).
- Підтримка роботи з сертифікатами: можливість зберігати, завантажувати та перевіряти сертифікати, видані центрами сертифікації.
- Інтеграція із зовнішніми системами: API для роботи в межах Web-сервісу.

2) Вибір варіанту бібліотеки

2.1 Огляд існуючих бібліотек

Для реалізації електронного цифрового підпису можуть застосовуватися дві групи бібліотек:

1. Бібліотеки для арифметики довільної точності, які надають базові можливості роботи з великими числами, необхідними у криптографії.
2. Бібліотеки криптографічних примітивів і протоколів, що безпосередньо реалізують алгоритми шифрування, підпису, генерації ключів та управління сертифікатами.

А) Бібліотеки арифметики

- GMP (GNU Multiple Precision Arithmetic Library) — одна з найбільш відомих і продуктивних бібліотек для роботи з числами довільної точності. Використовується як низькорівнева основа для криптографічних алгоритмів (RSA, ECC). Має високу швидкодію, підтримує оптимізації під різні архітектури, поширюється під ліцензією LGPL.

- MPIR (Multiple Precision Integers and Rationals) — форк GMP, орієнтований на Windows-платформи.

Пряме використання таких бібліотек для побудови Web-сервісу ЕЦП малоймовірне, однак вони можуть слугувати базою для криптографічних реалізацій.

Б) Бібліотеки криптографічних примітивів

- **OpenSSL (C)** — найпоширеніша бібліотека для роботи з криптографічними алгоритмами, сертифікатами X.509, протоколами TLS/SSL. Має вбудовану підтримку стандартів PKCS#7 (CMS), PKCS#12, роботу з сертифікатами та ключами. Підтримує FIPS-модуль для відповідності стандартам безпеки. Ліцензія Apache 2.0.
- **BouncyCastle (Java/.NET)** — кросплатформена криптобібліотека, яка підтримує широке коло алгоритмів, формати сертифікатів, реалізацію CMS (Cryptographic Message Syntax). Широко використовується у середовищі Java. Ліцензія MIT-подібна, є FIPS-сертифіковані версії.
- **libsodium (C)** — сучасна криптобібліотека, орієнтована на простоту і швидкість. Підтримує сучасні алгоритми (Ed25519, Curve25519), але не має нативної підтримки PKCS#7 чи X.509, що робить її менш зручною для побудови повноцінного Web-сервісу ЕЦП. Ліцензія ISC.
- **PyCryptodome (Python)** — бібліотека для мови Python, яка реалізує базові алгоритми шифрування, підпису та генерації ключів. Однак повноцінна робота з X.509 та PKCS#7 потребує додаткових модулів або інтеграції з OpenSSL. Ліцензія BSD.
- **Botan (C++)** — сучасна криптобібліотека з підтримкою багатьох алгоритмів, TLS, X.509 та інтеграції з PKCS#11. Проте реалізація CMS менш розвинена, ніж у OpenSSL чи BouncyCastle. Ліцензія BSD-2-Clause.

2.2 Критерії вибору бібліотеки

При виборі бібліотеки для реалізації електронного цифрового підпису у Web-сервісі необхідно враховувати такі критерії:

1. Безпека та відповідність стандартам
   - підтримка міжнародних стандартів (PKCS#7, PKCS#11, X.509, RFC 5652);
   - наявність сертифікації (наприклад, FIPS 140-2/3);
   - сучасність алгоритмів (RSA, ECDSA, Ed25519, SHA-2, SHA-3);
   - надійність реалізації, відсутність критичних вразливостей.
2. Продуктивність
   - ефективна реалізація криптоалгоритмів (швидке підписування та перевірка підпису);
   - можливість використання апаратних прискорень (AES-NI, AVX2, інструкції CPU);
   - масштабованість при обслуговуванні великої кількості запитів.

3. Зручність інтеграції з Web-сервісом
   - наявність API або CLI для інтеграції у веб-додатки;
   - підтримка різних мов програмування (C, Java, Python, .NET);
   - можливість роботи в контейнерах та хмарних середовищах.
4. Ліцензія
   - Open-source рішення з дозволом на комерційне використання (Apache, BSD, MIT) є пріоритетними для академічних і прикладних проєктів;
   - наявність комерційної підтримки та сертифікованих версій бажана для промислових рішень.

2.3 Обґрунтування вибору

Порівнявши можливості бібліотек, можна зробити висновки:

- libsodium та PyCryptodome зручні для простих задач криптографії, але не мають повноцінної підтримки стандартів PKI та CMS.
- Botan є перспективною бібліотекою для C++-систем, проте для реалізації Web-сервісу ЕЦП більше підходять зріліші рішення.
- BouncyCastle добре інтегрується у Java-середовище, але менш універсальна у випадку багатомовних проєктів.
- OpenSSL є найповнішим та найстабільнішим рішенням:
  - має вбудовану підтримку PKCS#7/CMS, X.509, PKCS#12;
  - підтримує інтеграцію з апаратними модулями (HSM) через PKCS#11;
  - сумісна з більшістю мов програмування та фреймворків;
  - ліцензується за Apache 2.0, що спрощує як навчальне, так і комерційне використання.

2.4 Вибрана бібліотека

Для реалізації Web-сервісу електронного цифрового підпису обираємо бібліотеку OpenSSL (версія 3.x).

Переваги вибору:

- повна підтримка стандартів PKCS#7 (CMS), X.509 та PKCS#11;
- наявність FIPS-модуля для підвищених вимог безпеки;
- висока продуктивність та оптимізація під сучасні процесори;
- активна спільнота та регулярні оновлення;
- зручність інтеграції в будь-який стек технологій (C, Python, Java через JNI, Node.js через модулі);
- ліцензія Apache 2.0, яка не накладає обмежень на використання.

Альтернатива: у випадку, якщо сервіс реалізується повністю у середовищі Java/Kotlin, доцільним вибором буде BouncyCastle, який надає нативний API для CMS та інтегрується як JCE-провайдер.

3) Формування технічних вимог

Технічні вимоги є основою для проектування та подальшої реалізації Web-сервісу електронного цифрового підпису (ЕЦП). Вони поділяються на функціональні та нефункціональні, причому перші описують, які завдання повинна виконувати система, а другі — які характеристики та якість її роботи необхідно забезпечити.

3.1 Функціональні вимоги

Створення пари ключів

- система повинна забезпечувати генерацію асиметричних ключових пар (RSA, ECDSA, Ed25519);
- користувач має мати можливість отримати відкритий ключ у стандартному форматі (X.509), а приватний ключ зберігати у захищеному контейнері (PKCS#12 або HSM).

Підписування даних

- Web-сервіс повинен надавати можливість підписування довільних даних (файлів, повідомлень, транзакцій);
- реалізація підпису здійснюється за допомогою приватного ключа з використанням алгоритмів, що відповідають сучасним стандартам (RSA з SHA-256, ECDSA, EdDSA);
- результатом є сформований підпис у форматі CMS/PKCS#7.

Перевірка підпису

- система повинна забезпечувати перевірку цифрового підпису за допомогою публічного ключа;
- перевірка включає зіставлення хеш-значення підписаних даних із розшифрованим підписом;
- необхідна валідація сертифікатів (перевірка ланцюга довіри, статусу відкликання через CRL/OCSP).

Підтримка сертифікатів

- Web-сервіс має працювати з сертифікатами стандарту X.509;
- повинна бути можливість завантаження та зберігання сертифікатів користувачів;
- передбачена підтримка експорту/імпорту сертифікатів у форматах DER, PEM, PKCS#12.

Логування операцій

- усі операції (створення ключів, підпис, перевірка, завантаження сертифікатів) мають фіксуватися в журналі подій;
- журнал повинен зберігати дату, час, тип операції та ідентифікатор користувача;
- доступ до логів має бути обмеженим, щоб уникнути витоку конфіденційних даних.

3.2 Нефункціональні вимоги

1. Продуктивність
   - час генерації цифрового підпису не повинен перевищувати 100 мс для документів розміром до 1 МБ;
   - перевірка підпису має виконуватися не довше, ніж за 200 мс при тих же умовах;
   - обробка великих файлів (>10 МБ) повинна здійснюватися потоково, без завантаження всього документа в пам’ять.
2. Масштабованість
   - система повинна підтримувати роботу з не менше 1000 одночасних користувачів;
   - архітектура має бути горизонтально масштабованою (через контейнеризацію та балансування навантаження).
3. Безпека
   - приватні ключі користувачів повинні зберігатися лише у зашифрованому вигляді або в апаратних модулях (HSM/TPM);
   - всі мережеві з’єднання мають захищатися протоколом TLS 1.2/1.3;
   - система повинна бути стійкою до основних атак: SQL-ін’єкцій, XSS, CSRF, Replay-атак, Man-in-the-Middle;
   - необхідно забезпечити багаторівневий контроль доступу (рольова модель, аудит дій користувачів).

4) Проектування Web-сервісу

4.1 Вибір архітектурного стилю

Для сервісу електронного цифрового підпису обрано REST API з такими міркуваннями:

- Простота інтеграції: REST-інтерфейси легко споживаються будь-якими клієнтами (браузер, мобільні застосунки, сервери) через HTTP(S).
- Формати даних: підтримка як `application/json`, так і бінарних форматів (наприклад, `application/pkcs7-signature`, `application/pkcs7-mime`), а також `multipart/form-data` для файлів.
- Масштабованість: REST добре працює за балансувальниками навантаження та в контейнеризованому середовищі.
- Спостережуваність: стандартні метрики/логування/трасування (OpenTelemetry) і звичні засоби кешування/обмеження швидкості.

SOAP розглядався як альтернатива для середовищ з вимогою XML/SOAP-політик і строгих контрактів WSDL, але для нашого випадку він дає зайву складність без суттєвих переваг.

4.2 Логічна архітектура та схема взаємодії

Складові системи:

1. API Gateway / Edge — термінує TLS, авторизує запити (OAuth2/JWT/API-Keys), робить rate limiting.
2. Signature Service (Core API) — REST-контролери: `/sign`, `/verify`, `/keys/*`, `/certs/*`. Інкапсулює бізнес-логіку.
3. Crypto Engine — обгортка над OpenSSL 3 (CLI або нативні виклики), робота з CMS/PKCS#7, X.509, CRL/OCSP.
4. Key Management
   - HSM/PKCS#11 (переважно) — приватні ключі ніколи не покидають апаратний модуль;
   - Software keystore (PKCS#12) — навчальний варіант/резерв (шифрування, MFA-доступ).
5. Certificate Store — сховище сертифікатів (видані/довірені CA, ланцюги, CRL/OCSP-налаштування).
6. Audit & Logging — незмінне логування операцій (підпис, верифікація, керування ключами), журнал дій адміністратора.
7. Monitoring — метрики (латентність підпису/верифікації, помилки), алерти.

Текстова схема послідовностей (узагальнено):

- Підпис
  1. **Клієнт** → `POST /v1/sign`: дані/хеш + ідентифікатор ключа/сертифіката
  2. API перевіряє токен доступу, права, ліміти
  3. Crypto Engine формує CMS SignedData, звертається до HSM через PKCS#11
  4. Повертається CMS (attached або detached), подія логуються в Audit
- Перевірка
  1. **Клієнт** → `POST /v1/verify`: CMS + (необов'язково) вихідні дані при detached
  2. API виконує перевірку підпису та ланцюга довіри (CRL/OCSP)
  3. Повертається результат (valid/invalid, причини невдачі), Audit подія

4.3 Основні ендпоїнти REST API (v1)

Нижче наведено мінімально необхідний набір. Всі ендпоїнти працюють по HTTPS, вимагають авторизацію (наприклад, `Authorization: Bearer <JWT>`), а відповіді містять уніфіковані помилки з кодами та машинночитними причинами.

Позначення:

CMS = Cryptographic Message Syntax (PKCS#7); attached = підпис із вкладеним вмістом; detached = підпис без вкладеного вмісту.

4.3.1 Генерація ключів

POST `/v1/keys/generate`

Призначення: створити пару ключів у HSM або у програмному сховищі (для лабораторії).

Запит (JSON):

```
{
  "algo": "RSA|ECDSA|Ed25519",
  "keySize": 2048,
  "curve": "secp256r1",
  "protection": "HSM|PKCS12",
  "label": "signing-key-01",
  "subject": "CN=Demo User,O=Org,C=UA",
  "exportCert": true
}
```

Відповідь 201 (JSON):

```
{
  "keyId": "k_abc123",
  "certId": "c_abc123",
  "publicKeyPem": "-----BEGIN PUBLIC KEY-----...",
  "certificatePem": "-----BEGIN CERTIFICATE-----..."
}
```

Примітки:

- Якщо `protection=HSM`, приватний ключ не експортується.
- Можлива асинхронна видача сертифіката через інтеграцію з CA (поза межами базового сценарію лабораторної).

4.3.2 Підпис даних

POST `/v1/sign`

Призначення: сформувати CMS-підпис (attached або detached).

Запит (JSON або multipart/form-data):

```
{
  "keyId": "k_abc123",
  "signMode": "attached|detached",
  "hashAlg": "SHA256",
  "content": "base64-encoded bytes"
}
```

або `multipart/form-data` з полем файлу `file`.

Відповідь 200 (JSON або PKCS7):

- Якщо JSON:

```
{
```

```
  "cms": "base64-encoded CMS",
  "format": "application/pkcs7-signature",
  "signMode": "attached"
}
```

- Або бінарна відповідь з `Content-Type: application/pkcs7-signature`

Примітки:

- Для `detached` замість `attached` повертається підпис без вкладення контенту.
- Опційно: параметр `tsaUrl` для додавання мітки часу (RFC 3161).

4.3.3 Підпис даних

POST `/v1/verify`

Призначення: перевірити CMS-підпис, ланцюг довіри, статус відкликання.

Запит (JSON або multipart/form-data):

```
{
  "cms": "base64-encoded CMS",
  "detachedContent": "base64-encoded bytes (optional)"
}
```

Відповідь 200 (JSON):

```
{
  "isValid": true,
  "signers": [
    {
      "subject": "CN=Demo User,O=Org,C=UA",
      "serialNumber": "01A2...",
      "alg": "sha256WithRSAEncryption",
      "time": "2025-09-20T13:45:00Z",
      "chainValid": true,
      "revocation": "good"
    }
  ],
  "warnings": []
}
```

Примітки:

- Для `detached` необхідно надсилати вихідні дані.
- Перевірка CRL/OCSP налаштовується в конфігурації сервісу.

4.3.4 Робота з сертифікатами

POST `/v1/certs/import`

Вхід: PEM/DER/PKCS#12; вихід: `certId`, метадані.

GET `/v1/certs/{certId}`
Повертає PEM/метадані, статус довіри.
GET `/v1/certs/chain/{certId}`
Повертає повний ланцюг довіри (за наявності).
DELETE `/v1/certs/{certId}`
Видалення (або позначення як недовірений у локальному сторі).

4.3.5 Керування ключами (адміністративне)

GET `/v1/keys/{keyId}` — метадані ключа (алгоритм, розташування: HSM/PKCS12).
POST `/v1/keys/rotate` — ротація ключа з перевипуском сертифіката.
POST `/v1/keys/disable` — тимчасово заборонити використання ключа.
У продакшн-сценаріях ці операції мають додаткові контролі доступу (MFA, 4-eyes principle).

4.3.6 Журнали та спостережуваність

GET `/v1/audit/logs?from=...&to=...&actor=...` — вибірка подій (ролеобмежений доступ).
GET `/v1/health` — стан сервісу (liveness/readiness).
GET `/v1/metrics` — метрики у форматі Prometheus.

4.4 Нотатки з безпеки та експлуатації

- Транспортний рівень: лише HTTPS (TLS 1.2/1.3), сучасні шифрнабори, HSTS.
- Аутентифікація/авторизація: OAuth2/OIDC (JWT), рольова модель (admin, signer, verifier, auditor).
- Ключі: за замовчуванням у HSM через PKCS#11; у лабораторному середовищі допускається PKCS#12 з сильним паролем і KMS-зашифруванням.
- Аудит: незмінний, із часовими мітками, кореляцією `requestId` та `actorId`.
- Обмеження навантаження: rate limiting, circuit breaker, черги для «важких» завдань.
- Версіонування API: префікс `/v1`, сумісність у межах мінорних оновлень.

4.5 Приклади використання (фрагменти cURL)

Підпис (attached):

```
curl -X POST https://sign.example.com/v1/sign \
  -H "Authorization: Bearer <TOKEN>" \
  -H "Content-Type: application/json" \
  -d '{
        "keyId": "k_abc123",
        "signMode": "attached",
        "hashAlg": "SHA256",
```

```
        "content": "<base64>"
      }' -o signed.p7s
```

Перевірка:

```
curl -X POST https://sign.example.com/v1/verify \
  -H "Authorization: Bearer <TOKEN>" \
  -H "Content-Type: application/json" \
  -d '{
        "cms": "<base64-of-p7s>"
      }'
```

5) Обґрунтування вибору

5.1 Чому обрано саме OpenSSL 3.x

Вимоги до нашого веб-сервісу ЕЦП включають роботу зі стандартами PKI (X.509), підтримку CMS/PKCS#7 для юридично значущих підписів, можливість інтеграції з HSM через PKCS#11, надійність і продуктивність. Серед розглянутих бібліотек саме OpenSSL 3.x найповніше і «з коробки» закриває ці потреби, а також надає зрілий інструментарій для експлуатації в реальних системах.

Ключові аргументи:

1. Повний стек стандартів OpenSSL забезпечує нативну підтримку CMS/PKCS#7, X.509, PKCS#12, CRL/OCSP-перевірок та ін. Це дозволяє формувати і валідувати підписи у форматах, що прийняті в електронному документообігу, без додаткових прошарків.
2. Сумісність із апаратним захистом ключів Модель provider у версіях 3.x дає можливість підключати PKCS#11-постачальників і працювати з ключами на токенах/HSM так, щоб приватний ключ ніколи не залишав захищений модуль. Це критично для виробничих сценаріїв і корисно навіть у навчальному проєкті (правильна архітектурна основа).
3. Продуктивність і зрілість реалізацій OpenSSL оптимізований під сучасні CPU та інструкції, має багаторічну історію аудиту і використання в інфраструктурах великих масштабів. Для нашого сервісу це означає стабільну швидкодію підпису/перевірки та передбачувану поведінку під навантаженням.
4. Інтеграційна гнучкість

   Є два перевірені шляхи інтеграції:

   - CLI утиліти (`openssl cms`, `openssl x509` тощо) — швидкий старт і простий виклик з будь-якої мови;
   - біндінги/обгортки — для тіснішої інтеграції у вибраний стек (Python/Go/Node.js тощо).

     Це зручно в умовах лабораторної: можна почати з CLI, а потім мігрувати на нативні виклики.

5. Ліцензійна прозорість і екосистема: Ліцензія Apache 2.0 підходить як для навчальних, так і для комерційних цілей. Навколо OpenSSL існує велика кількість посібників, прикладів, контейнерних імеджів і best practices для деплойменту та моніторингу.

Висновок: OpenSSL 3.x забезпечує необхідний баланс між функціональністю PKI/CMS, безпекою, продуктивністю та простотою інтеграції. Для стеку Java/Kotlin рівнозначною альтернативою за покриттям CMS є BouncyCastle; однак з огляду на технологічну нейтральність проєкту базовим вибором лишається OpenSSL.

5.2 Порівняння з альтернативами (переваги/недоліки)

Бібліотека	Переваги	Недоліки	Висновок щодо придатності
OpenSSL (C, v3.x)	Повна підтримка CMS/PKCS#7, X.509, PKCS#12; інтеграція з HSM/PKCS#11; висока продуктивність; зрілі інструменти CLI; широка екосистема; Apache 2.0	Низькорівневий C-API складніший за high-level бібліотеки; робота з CLI потребує акуратної обробки помилок і безпечного управління тимчасовими файлами	Оптимальний вибір для веб-сервісу ЕЦП у багатомовному середовищі
BouncyCastle (Java/.NET)	Натівна CMS для JVM/.NET; інтеграція через JCE/JCA; хороше покриття алгоритмів; наявні FIPS-варіанти; дружня ліцензія	Орієнтована переважно на JVM/.NET; за межами цих стеків потребує додаткових шарів	Відмінний вибір для Java/Kotlin/.NET сервісів; у нашому нейтральному стеку — альтернатива
libsodium (C)	Сучасні швидкі примітиви (Ed25519/Curve25519), простий та безпечний API, висока продуктивність, ISC	Немає CMS/X.509/PKCS#7; непридатна для класичного юридично значущого ЕЦП без додаткових шарів	Підійде для «сучасних» схем підпису без PKI; не підходить для нашого ЕЦП-сервісу «з коробки»

PyCryptodome (Python)	Зручна для Python, базові примітиви RSA/ECC/AES, BSD; швидкий прототипінг	Немає повної CMS/X.509 із коробки; для PKI все одно доведеться викликати OpenSSL або збирати ASN.1 вручну	Гарний допоміжний шар у Python; не самодостатня для PKI/CMS
Botan (C++)	Багатий набір алгоритмів, TLS, X.509, PKCS#11 підтримка; BSD-2-Clause; приємний C++-API	Реалізація CMS поступається OpenSSL/BC; менша «впізнаваність» у DevOps-екосистемі	Може бути вибором для C++-проєктів; менш бажана для CMS-орієнтованого сервісу
GMP / MPIR (bignum)	Максимальна швидкодія для великої арифметики; корисно для кастомних реалізацій	Це не PKI/підпис — немає CMS/X.509, немає високорівневих протоколів	Не відповідає завданню як основа для веб-сервісу ЕЦП

5.3 Узгодження вибору з вимогами проєкту

Вимога	Як покривається OpenSSL
Підпис/перевірка у форматі CMS/PKCS#7	Команда `openssl cms -sign/-verify`, або нативні виклики через API
Підтримка сертифікатів X.509, CRL/OCSP	`openssl x509, openssl ocsp`, перевірка ланцюга довіри
Робота з HSM (PKCS#11)	Модель provider (v3) + підключення PKCS#11-провайдерів; приватний ключ не покидає HSM
Продуктивність	Оптимізації під сучасні CPU, перевірена швидкодія у великих інсталяціях
Зручна інтеграція з веб-сервісом	Вибір: CLI (швидкий старт) або біндінги; легко контейнеризується
Ліцензія	Apache 2.0 — підходить для навчального та промислового використання

5.4 Ризики й шляхи їх мінімізації

- Складність низькорівневого API (C): Спочатку використовувати CLI-інтеграцію (`openssl cms`), а далі — поступово перейти на стабільні обгортки, суворо валідуючи вхід/вихід.
- Конфігурація PKI (ланцюги довіри, CRL/OCSP): Стандартизувати конфіг-файли, прописати політики валідації, додати інтеграційні тести з тестовими CA/CRL/OCSP.
- Операційні помилки при роботі з ключами/файлами: Ізолювати тимчасові файли, використовувати безпечні директорії, вмикати докладний audit log, застосувати «4-очі» та MFA для адмін-операцій.

Висновок: З урахуванням вимог (CMS/PKCS#7, X.509, PKCS#11/HSM, продуктивність, інтеграційна гнучкість, ліцензія) OpenSSL 3.x є найбільш збалансованим вибором для побудови веб-сервісу ЕЦП у нейтральному технологічному стеку. BouncyCastle залишається сильною альтернативою для стеків Java/Kotlin/.NET. Інші розглянуті бібліотеки або не надають необхідної підтримки PKI/CMS, або потребують значних доробок, що виходять за межі лабораторної роботи.

6) Висновки

6.1 Узагальнення результатів роботи

У ході виконання лабораторної роботи було проведено аналіз предметної області електронного цифрового підпису (ЕЦП), сформульовано функціональні та нефункціональні вимоги, а також спроєктовано архітектуру веб-сервісу для його реалізації.

Було визначено, що для ефективної роботи сервіс має підтримувати:

- генерацію криптографічних ключів;
- підписування та перевірку даних у стандартних форматах (CMS/PKCS#7);
- роботу із сертифікатами X.509, включаючи перевірку ланцюга довіри;
- зручне логування та аудит операцій.

Нефункціональні вимоги орієнтовані на продуктивність, масштабованість та безпеку, що є ключовими факторами для впровадження подібних рішень у реальні IT-системи.

Після огляду бібліотек для реалізації криптографічних операцій було обрано OpenSSL 3.x, оскільки вона забезпечує найширше покриття стандартів (PKCS#7, PKCS#11, X.509), має зрілу реалізацію та активну підтримку спільноти.

6.2 Відповідність технічних вимог завданням веб-сервісу ЕЦП

- Захищеність даних та відповідність стандартам — виконуються завдяки використанню перевірених криптографічних алгоритмів та протоколів.

- Надійність та масштабованість — архітектура REST API із можливістю горизонтального масштабування забезпечує підтримку великої кількості користувачів.
- Інтеграційна гнучкість — OpenSSL дозволяє легко інтегрувати сервіс у різні середовища (CLI, API, біндінги для різних мов).
- Юридична значущість — підтримка CMS/PKCS#7 та сертифікатів X.509 відповідає вимогам до електронного документообігу.

Таким чином, сформульовані вимоги повністю узгоджуються з поставленими завданнями розробки веб-сервісу ЕЦП.

6.3 Рекомендації для практичної реалізації

1. Реалізувати базовий прототип сервісу на основі REST API з ендпоїнтами `/sign`, `/verify`, `/keys/generate`.
2. Інтегрувати з OpenSSL через CLI або нативні бібліотеки, забезпечивши безпечне управління ключами.
3. Додати аудит і моніторинг: усі операції повинні логуватися із зазначенням часу, користувача та типу дії.
4. Забезпечити безпеку приватних ключів — для лабораторних умов достатньо PKCS#12 із паролем, у промислових — інтеграція з HSM/PKCS#11.
5. Протестувати масштабованість — використати інструменти навантажувального тестування (JMeter, k6) для перевірки стабільності.
6. Передбачити подальші розширення:
    - підтримка додаткових алгоритмів (EdDSA, ГОСТ);
    - інтеграція з Time Stamping Authority (TSA) для підписів з міткою часу;
    - додавання веб-інтерфейсу для зручності користувачів.

6.4 Підсумок

Розроблені технічні вимоги та архітектура веб-сервісу ЕЦП дозволяють створити ефективний, безпечний та масштабований інструмент, придатний як для навчальних цілей, так і для реальної експлуатації у системах електронного документообігу, банківських сервісах та онлайн-автентифікації.