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

ЗВІТ З ЛАБОРАТОРНОЇ РОБОТИ №2

З дисципліни «Методи реалізації криптографічних механізмів»

«ДОСЛІДЖЕННЯ ОСОБЛИВОСТЕЙ РЕАЛІЗАЦІЇ ІСНУЮЧИХ ПРОГРАМНИХ СИСТЕМ, ЯКІ ВИКОРИСТОВУЮТЬ КРИПТОГРАФІЧНІ МЕХАНІЗМИ ЗАХИСТУ ІНФОРМАЦІЇ»

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

КИЇВ – 2025

Мета роботи: Отримання практичних навичок побудови гібридних криптосистем.
Завдання: дослідити основні задачі, що виникають при програмній реалізації криптосистем. Запропонувати методи вирішення задачі контролю доступу до ключової інформації, що зберігається в оперативній пам'яті ЕОМ для різних (обраних) операційних систем.
Запропонувати методи вирішення задачі контролю правильності функціонування програми криптографічної обробки інформації. Порівняти з точки зору вирішення цих задач інтерфейси Crypto API, PKCS 11.
Підгрупа 1В. Реалізація для User Endpoint terminal.

Хід роботи

Для виконання даної лабораторної роботи було розроблено наступний план:

1. Розглянути теоретичні основи криптосистем, їх компоненти та ключову інформацію.
2. Проаналізувати проблеми та загрози, що виникають при зберіганні ключової інформації в оперативній пам'яті ЕОМ.
3. Запропонувати методи контролю доступу до ключів для різних операційних систем у середовищі користувацького терміналу (User Endpoint Terminal).
4. Розглянути методи забезпечення правильності та коректності функціонування програм криптографічної обробки інформації.
5. Провести порівняльний аналіз інтерфейсів CryptoAPI та PKCS#11 з точки зору вирішення задач контролю доступу до ключової інформації та перевірки коректності криптографічних операцій.
6. Зробити узагальнюючі висновки щодо ефективності застосованих методів та інтерфейсів.

1) Теоретичні основи криптосистем, їх компоненти та ключова інформація

Криптографічна система (криптосистема) — це сукупність алгоритмів, протоколів та засобів, призначених для забезпечення конфіденційності, цілісності, автентичності та доступності інформації. Основною метою криптосистем є перетворення відкритих даних у вигляд, що унеможливлює їх несанкціоноване прочитання чи модифікацію, а також відновлення початкового вигляду інформації за наявності дозволу.

До складу будь-якої криптосистеми входять такі основні компоненти:

1. Алгоритм шифрування та розшифрування – математичні процедури, що визначають правила перетворення відкритих даних (plaintext) у зашифровані (ciphertext) та навпаки.
2. Ключова інформація (криптографічні ключі) – набір параметрів (бітових послідовностей), необхідних для виконання операцій шифрування, розшифрування, підпису або перевірки цифрового підпису.
3. Протоколи обміну ключами – процедури, що забезпечують захищену передачу ключів між учасниками обміну.
4. Механізми автентифікації та контролю доступу – дозволяють перевіряти достовірність сторін та обмежувати доступ до криптографічних операцій.
5. Служби управління ключами – включають генерацію, зберігання, оновлення, ротацію, резервування та знищення ключів.

Особливе значення у криптосистемах має ключова інформація, оскільки саме ключ забезпечує секретність криптографічних перетворень. Навіть якщо алгоритм шифрування є загальнодоступним, без наявності ключа неможливо виконати розшифрування або підробку криптографічних даних. Тому ключі вважаються найціннішим елементом криптосистеми і потребують підвищеного рівня захисту.

Криптографічні ключі можуть бути:

- Симетричні — один і той же ключ використовується як для шифрування, так і для розшифрування інформації.
- Асиметричні — використовуються два пов’язані ключі: відкритий (public key) та закритий (private key). Відкритий ключ може бути доступним широкому колу користувачів, тоді як приватний має зберігатися у суворій секретності.

Процес забезпечення безпеки криптосистеми значною мірою залежить від правильності та надійності управління ключами, оскільки компрометація ключа веде до повної втрати захищеності інформації, незалежно від стійкості використовуваних алгоритмів.

2) Проблеми та загрози, що виникають при зберіганні ключової інформації в оперативній пам’яті ЕОМ

Під час функціонування криптографічних систем криптографічні ключі у певні моменти повинні бути доступними програмі для виконання операцій шифрування, розшифрування, підпису або перевірки підпису. З цієї причини ключі часто завантажуються та тимчасово зберігаються в оперативній пам’яті (RAM). Однак оперативна пам’ять не є захищеним середовищем зберігання, що створює низку потенційних загроз для безпеки криптосистеми.

Основні ризики, пов’язані зі зберіганням ключової інформації в оперативній пам’яті, полягають у наступному:

1. Атаки шляхом читання дампів пам’яті (Memory Dump Attack).
   Якщо зловмисник отримує доступ до фізичної або віртуальної пам’яті процесу, він може відновити криптографічні ключі у відкритому вигляді. Це можливо як у режимі реального часу (через відладчики та інструменти зчитування процесів), так і шляхом аналізу файлів дампів пам’яті.
2. Атака «холодного перезапуску» (Cold Boot Attack).
   При вимкненні живлення оперативна пам’ять не очищується миттєво. Дослідження показали, що дані в RAM можуть зберігатися від кількох секунд до хвилин. Використовуючи спеціальне обладнання, зловмисник може вилучити пам’ять і відновити ключову інформацію.
3. Атаки з використанням шкідливого програмного забезпечення.
   У разі зараження системи троянами або руткітами зловмисник може перехоплювати ключі безпосередньо під час їх використання додатком. Це особливо небезпечно на робочих станціях кінцевих користувачів (User Endpoint Terminal).
4. Атаки через інтерфейс налагодження (Debugging / Hooking).
   Доступ до процесу за допомогою налагоджувача дозволяє переглядати вміст регістрів та пам’яті, що може призвести до компрометації ключів, якщо програма не блокує можливість її відлагодження.
5. Некоректне видалення ключів з пам’яті.
   Якщо після завершення криптографічних операцій ключі не

видаляються або не перезаписуються, вони залишаються в пам’яті, що підвищує ризик їх подальшого відновлення.

Таким чином, оперативна пам’ять є потенційно вразливим середовищем для зберігання ключової інформації, і саме на етапі управління пам’яттю закладається значна частина безпеки криптографічних програм. Забезпечення захисту ключів вимагає застосування спеціальних методів, що мінімізують ризик витоку ключів як під час їх використання, так і після завершення криптографічних операцій.

3) Запропоновані методи контролю доступу до ключової інформації в оперативній пам’яті для різних операційних систем (User Endpoint Terminal)

Контроль доступу до ключової інформації в оперативній пам’яті кінцевого користувача (User Endpoint Terminal) має вирішальне значення для забезпечення конфіденційності криптографічних операцій. Нижче наведено набір практичних методів і механізмів, адаптованих під тріаду найбільш поширених операційних систем — Windows, Linux та macOS — а також загальні рекомендації для крос-платформної реалізації.

Загальні принципи (крос-платформенно)

1. **Мінімізація часу зберігання в явному вигляді** — ключі повинні перебувати в оперативній пам’яті лише протягом фактичного виконання операції; після завершення — негайна перезапис/знищення (zeroization).
2. **Мінімізація привілеїв процесу** — застосунок має працювати з найменшими необхідними правами (principle of least privilege).
3. **Використання апаратного захисту, якщо доступний** — HSM, TPM, Secure Enclave або інші апаратні модулі для зберігання або операцій над ключами.
4. **Запобігання дампам та відлагодженню** — вимкнення core dumps, виявлення/обмеження доступу через debug/ptrace, захист від інжекції/хукінгу.
5. **Пам’ять, що не підлягає підкачуванню (no-swap)** — блокування ключових буферів у фізичній пам’яті (mlock/VirtualLock) для запобігання запису у swap/файли підкачки.

6. **Контроль цілісності процесу та коду** — підписування бінарників, перевірки контрольних сум, обмеження дозволів на виконання та модифікацію.
7. **Аудит та логування доступу до криптоопераций** — без запису ключів у логи; реєстрація фактів доступу/помилок.
8. **Захищені міжпроцесні канали (IPC)** — якщо ключ передається процесам, використовувати захищені та аутентифіковані IPC (secured sockets, named pipes з аутентифікацією).

Windows (User Endpoint Terminal)

1. **Використання системних сховищ та API**
    - Зберігати ключі у захищених сервісах або використовувати апаратні токени/смарткартки через PKCS#11 чи CAPI/CNG.
    - Для тимчасового шифрування/захисту в пам’яті застосовувати `CryptProtectMemory / BCryptProtectMemory` або DPAPI для персистентного збереження, з урахуванням вимог до доступності на машині користувача.
2. **Блокування пам’яті та запобігання підкачуванню**
    - Використовувати `VirtualLock` для блокування сторінок з ключами у фізичній пам’яті.
    - Вимкнути або обмежити створення дампів (`SetErrorMode`, налаштування політик створення core dumps), встановити політику `SetProcessMitigationPolicy` для зменшення вразливостей.
3. **Захист від відлагодження та інжекцій**
    - Перевіряти наявність відлагоджувача (`IsDebuggerPresent`) та використовувати апаратні політики захисту процесу (Process Mitigation Policies).
    - Використовувати підписування коду та AppLocker/Windows Defender Application Control для запобігання запуску неавторизованих модифікацій.
4. **Hardware-backed ключі**
    - Використовувати TPM або Windows Hello / Trusted Platform Module для зберігання приватних ключів або виконання

криптоопераций без вивантаження закритого ключа в загальнодоступну пам’ять.

Linux (User Endpoint Terminal)

1. **Блокування пам’яті і відключення підкачування**
    - Використовувати `mlock()` / `mlockall()` для блокування пам’яті; застосовувати `MADV_DONTFORK`, `mprotect()` для контролю доступу.
    - Забезпечити, щоб процес не створював core dump (через `setrlimit(RLIMIT_CORE, 0)`), встановити `prctl(PR_SET_DUMPABLE, 0)`.
2. **Ядрові механізми й політики доступу**
    - Використовувати Linux Kernel Keyring (sic), якщо це відповідає моделі загрози, або інтегруватися з PKCS#11 токенами.
    - Надавати додаткові обмеження через LSM (SELinux / AppArmor) — задавати політики, що обмежують доступ процесу до ресурсів, файлової системи і можливості ptrace.
3. **Запобігання ptrace / debug**
    - Забороняти іншим процесам прикріплюватися (ptrace) — `prctl(PR_SET_PTRACER, PR_SET_PTRACER_ANY)` з обережністю; використовувати механізми, що обмежують ptrace до привілейованих процесів.
4. **Сандбоксування та розділення привілеїв**
    - Використовувати namespaces, seccomp-filters для обмеження системних викликів, працювати як непривілейований користувач.
    - Запускати криптомодуль у відокремленому процесі з чіткими IPC та аутентифікацією (socketpair + credentials).
5. **Управління ключами**
    - Підключення апаратних модулів через PKCS#11 або використання спеціалізованих демонов (наприклад, gnome-keyring, but with caution for user endpoints) — перевага при наявності HSM/TPM.

macOS (User Endpoint Terminal)

1. **Використання Keychain та Secure Enclave**
    - Для зберігання та керування ключами рекомендується використовувати системний Keychain; де можливо — апаратну ізоляцію через Secure Enclave (наприклад, для приватних ключів, що підтримуються апаратно).
    - Обмежувати доступ через TCC (Transparency, Consent, and Control) та SIP (System Integrity Protection) для зменшення атак на платформу.
2. **Блокування пам'яті та управління дампами**
    - Використовувати `mlock()`/`mlockall()` та `vm_protect()` для захисту буферів з ключами; заборонити core dumps через системні параметри.
    - Переконатися, що ключі не виводяться у логи і не опиняються в crash reports.
3. **Контроль над відлагодженням та цілісністю коду**
    - Використовувати підписування коду (code signing), перевірку підпису під час виконання та App Sandbox для обмеження діяльності процесу.
    - Використовувати механізми перевірки цілісності (notarization) для дистрибуції бінарників.

Практичні патерни для User Endpoint Terminal

1. **Аппаратно-підтримувані операції**
    - По можливості переміщувати критичні операції (підпис, розшифрування) в апаратний модуль (TPM, Secure Enclave, HSM), щоб закритий ключ ніколи не з'являвся у загальнодоступній RAM у відкритому вигляді.
2. **Короткоживучі сесійні ключі**
    - Використовувати короткочасні сесійні ключі, отримані від довготривалого ключа через KDF; довготривалі ключі зберігати в захищеному сховищі, а в пам'яті робочі ключі — мінімізувати час життя.
3. **Захищене введення/аутентифікація**

    - Захищати доступ до ключового сховища PIN/біометрією; забезпечити обмеження кількості спроб та блокування після неуспішних спроб.
4. **Zeroization та перезапис**
    - Після використання ключі та їх копії (включаючи стек-буфери) повинні бути негайно перезаписані випадковими даними і знеособлені; бібліотеки повинні надавати гарантовані механізми очищення.
5. **Постійний аудит та самоперевірка**
    - Вбудувати self-tests (перевірки коректності криптоопераций) під час ініціалізації, контроль цілісності бінарника, та логування подій доступу без розкриття секретів.
6. **Використання перевірених бібліотек/стеків та оновлення**
    - Використовувати перевірені криптографічні бібліотеки (з офіційною підтримкою) та оперативно застосовувати патчі безпеки; уникати самописних реалізацій криптографії.

Ефективний контроль доступу до ключової інформації на User Endpoint Terminal досягається поєднанням апаратних засобів (HSM/TPM/Secure Enclave), належного програмного управління пам’яттю (mlock/VirtualLock, zeroization), обмеження привілеїв та політик ОС (LSM, App Sandbox, SIP, політики відлагодження) і практик безпечної розробки (аудит, підписування коду, оновлення). Для кожної ОС існують специфічні API й інструменти, але загальна стратегія — мінімізувати присутність відкритих ключів у RAM, захистити середовище виконання та використовувати апаратні можливості за наявності — залишається однаковою.

4) Методи забезпечення правильності та коректності функціонування програм криптографічної обробки інформації

Забезпечення правильності та коректності функціонування програм криптографічної обробки інформації є ключовим елементом побудови безпечних інформаційних систем. Навіть за наявності стійких алгоритмів криптографічний захист може бути порушений через помилки в реалізації, неправильне використання ключів або некоректне управління пам’яттю. Тому важливо застосовувати комплекс методів перевірки коректності та контролю роботи криптографічних програм.

Основні методи забезпечення коректності криптографічних обчислень включають:

1. **Використання тестових векторів (Test Vectors).**
   Стандарти NIST, ISO, RFC та інші організації розробляють еталонні входи та відповідні криптографічні результати. Перевірка реалізації на таких наборах дозволяє упевнитися у відповідності алгоритму специфікації.
   Наприклад, для алгоритмів AES, SHA-2, RSA існують офіційні тестові набори, які повинні бути коректно відпрацьовані програмою.
2. **Самотестування (Self-tests) під час ініціалізації.**
   Багато сертифікованих криптобібліотек (наприклад, OpenSSL FIPS, BoringSSL) виконують вбудоване самотестування при завантаженні. Це дозволяє виявити:
   - пошкодження коду (биті або модифіковані бібліотеки),
   - помилки апаратної реалізації,
   - некоректні параметри ключів.
3. **Перевірка цілісності виконуваних модулів.**
   Застосовується контроль хеш-сум або цифрових підписів бібліотек, модулів і конфігурацій.
   Це запобігає атакам заміни (substitution attacks), коли зловмисник підміняє криптографічні функції модифікованими.
4. **Контроль правильності ключової інформації.**
   Перед виконанням операцій шифрування/розшифрування перевіряється:
   - довжина ключа,
   - відповідність формату відкритого та закритого ключів,
   - відсутність нульових або слабких ключів (наприклад, ключів із низькою ентропією).
5. **Застосування сертифікованих криптографічних модулів.**
   Використання бібліотек, що пройшли сертифікацію за стандартами **FIPS 140-2/140-3**, знижує ризики, пов’язані з некоректними реалізаціями алгоритмів. Такі модулі містять:
   - механізми самотестування,
   - захист пам’яті,
   - контроль і аварійне завершення при виявленні помилок.
6. **Механізми виявлення збоїв під час роботи (Runtime Checks).**
   До них належать:

- перевірка коректності форматів даних після криптоопераций,
- контроль повторного використання одноразових параметрів (nonce, IV),
- перевірка коректності заповнення блоків перед розшифруванням.

7. **Аудит криптографічних операцій.**
   Ведення журналів з реєстрацією:
   - фактів використання ключів,
   - часу виконання операцій,
   - аномалій у роботі.
     При цьому важливо, щоб самі ключі не потрапляли в журнали.

Коректність криптографічної програми забезпечується як на етапі розробки (використання еталонних тестів, сертифікованих бібліотек), так і під час виконання (самоконтроль, перевірка параметрів, аудит). Недостатній контроль правильності роботи криптографічних функцій може призвести до компрометації системи, навіть якщо алгоритми та ключі є криптографічно стійкими.

5) Порівняльний аналіз інтерфейсів CryptoAPI та PKCS#11 щодо контролю доступу до ключової інформації та забезпечення коректності криптографічних операцій

Для реалізації криптографічних механізмів у сучасних інформаційних системах широко використовуються стандартні інтерфейси прикладного програмування (API), які надають засоби для управління ключами, шифрування, розшифрування, формування та перевірки цифрових підписів. Найбільш розповсюдженими серед них є CryptoAPI (Microsoft) та PKCS#11 (стандарт, розроблений RSA Laboratories). Вони суттєво відрізняються за архітектурою, моделлю управління ключами та підходами до забезпечення безпеки.

CryptoAPI

CryptoAPI є власним криптографічним інтерфейсом Windows і забезпечує інтеграцію з інфраструктурою CSP (Cryptographic Service Providers) та CNG (Cryptography Next Generation). Цей інтерфейс дозволяє:

- зберігати приватні ключі у захищених контейнерах Windows,

- використовувати апаратні модулі (TPM, смарт-картки) через провайдерів,
- здійснювати перевірку цілісності бібліотек та підписування коду.

Перевагою CryptoAPI є глибока інтеграція із системними механізмами контролю доступу Windows (ACL, UAC, Active Directory), що забезпечує централізоване управління правами на використання ключів. Коректність криптографічних операцій контролюється через вбудовані механізми самотестування і сертифікації відповідно до стандартів FIPS 140-2/3.

PKCS#11

PKCS#11 є крос-платформним інтерфейсом, який стандартизує роботу з криптографічними токенами, USB-ключами, HSM та іншими пристроями захищеного зберігання ключів. Основна модель PKCS#11 базується на розмежуванні доступу до ключів через PIN-коди, сесії та атрибути об’єктів.

PKCS#11 дозволяє виконувати криптоопераці без вилучення приватного ключа з токена, що знижує ризик появи ключів в оперативній пам’яті у відкритому вигляді. Контроль коректності операцій забезпечується через визначені механізми сесійного доступу та апаратні самотести HSM

Порівняльна таблиця CryptoAPI та PKCS#11

Критерій порівняння	**CryptoAPI (Windows)**	**PKCS#11 (крос-платформенно)**
Платформа	Тісно інтегрований з Windows	Працює на Windows, Linux, macOS
Модель управління ключами	Контейнери ключів у Windows + TPM	Токени/модулі (USB, HSM, смарт-картки)

Контроль доступу до ключів	ACL, політики груп, Active Directory	PIN-коди, ролі та атрибути об’єктів токена
Зберігання ключів у пам’яті	Можливе зберігання у RAM, але захист через DPAPI та VirtualLock	Ключ не покидає токен — операції виконуються всередині пристрою
Підтримка апаратного захисту	TPM, Smart Card через провайдери	HSM, Smart Card, TPM через універсальний драйвер
Механізми самоперевірки	Вбудовані FIPS self-tests	Апаратні self-tests HSM / токенів
Зручність інтеграції	Висока при розробці під Windows	Зручний для крос-платформних систем та стандартів
Гнучкість	Обмежена платформою	Висока, модульна і розширювана модель

CryptoAPI є оптимальним рішенням у середовищах Windows, де важливе централізоване управління доступом, інтеграція з корпоративними службами та використання засобів безпеки ОС.
PKCS#11 є більш універсальним і безпечнішим у контексті захисту ключів, оскільки дозволяє виконувати криптографічні операції без прямого доступу до приватного ключа та підтримує широкий спектр апаратних засобів захисту.

Таким чином, при виборі інтерфейсу для реалізації криптосистем у середовищі User Endpoint Terminal слід враховувати:

- платформу та середовище розгортання,
- вимоги до захисту ключів,
- наявність або відсутність апаратних засобів безпеки.

6) Висновки

У ході виконання лабораторної роботи було проаналізовано особливості програмної реалізації криптографічних систем, зокрема питання захисту ключової інформації та забезпечення коректності криптографічних операцій. Дослідження показало, що найбільш вразливим елементом криптосистем є саме криптографічний ключ, оскільки його компрометація призводить до повної втрати криптографічної стійкості незалежно від обраного алгоритму. Зберігання ключів в оперативній пам’яті створює загрозу несанкціонованого доступу через дампи пам’яті, атаки холодного перезапуску, шкідливе ПЗ та засоби відлагодження.

Розглянуті методи захисту ключової інформації показали, що ефективний захист досягається за рахунок поєднання програмних і апаратних засобів. Зокрема, блокування пам’яті, запобігання її підкачуванню, обмеження доступу через механізми ядра та використання апаратних модулів (TPM, Secure Enclave, HSM) дозволяють значно зменшити ймовірність витоку ключів у середовищі користувацького терміналу (User Endpoint Terminal). Важливим також є своєчасне очищення пам’яті (zeroization) та використання захищених протоколів доступу до ключів.

Порівняльний аналіз інтерфейсів CryptoAPI та PKCS#11 показав, що вони реалізують різні підходи до контролю доступу та зберігання ключів. CryptoAPI забезпечує тісну інтеграцію з Windows і централізоване управління правами доступу, що зручно для корпоративних середовищ. PKCS#11, у свою чергу, є універсальнішим та апаратно-орієнтованим рішенням, яке дозволяє виконувати криптографічні операції без передачі закритого ключа у RAM, що підвищує рівень безпеки в мультиплатформенних системах.

Отже, оптимальний вибір методів та інтерфейсів залежить від середовища функціонування криптосистеми, вимог до рівня захисту та наявності апаратних засобів безпеки. Для систем класу User Endpoint Terminal найбільш доцільним є комбінований підхід, який поєднує механізми захисту пам’яті, мінімізацію часу перебування ключів у відкритому вигляді

та використання апаратних модулів для зберігання або виконання криптографічних операцій.