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

Звіт з виконання комп'ютерного практикуму

# ДОСЛІДЖЕННЯ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ КРИПТОСИСТЕМИ. ІНТЕРФЕЙС PKCS11

Виконали студенти
групи ФІ-32мн
Величко Олена,
Мельник Ілля,
Міснік Аліна

Перевірили:
Селюх П.В.

Київ — 2023

**Мета роботи:** дослідити основні задачі, що виникають при програмній реалізації криптосистем. Запропонувати методи розв'язання задачі контролю доступу до ключової інформації, що зберігається в оперативній пам'яті ЕОМ для токена чи смарткарти.

**Постановка задачі:** Розглянути можливості реалізації програмної криптосистеми. Розглянути можливість використання PKCS11 та його інтерфейсу.

# 1 ХІД РОБОТИ

## 1.1 Реалізація криптосистеми

Реалізація криптосистеми потребує комплексного підходу. Це ряд вимог, які перш за все повинен перед собою поставити споживач або компанія, що збирається надалі використовувати її. Перш за все необхідно поставити ціль, для чого саме вам необхідна криптосистема. Якщо це малі токени, банківські картки, то ми повинні забезпечити швидкодію системи з ними, використовувати швидкі і надійні функції шифрування/дешифрування. Якщо наша інформація не є надто цінною, то можна не вкладати великі гроші у безпеку, оскільки як ми знаємо, надійні криптосистеми з великими обчисленнями вартують великих грошей. Або, чи взагалі потрібно нам шифрування? Чи підпис? Ставлячи чи запитання, ми можемо конкретно сформувати функції, що необхідні для нашої криптосистеми.

Припустимо, ми маємо розуміння, для чого нам криптосистема, що далі? Наступним кроком буде важка робота з реалізації основних функцій та властивостей, яким повинна задовольняти криптосистема. Наведемо їх нижче:

1) **Безпека алгоритмів**. До вибору алгоритму, або як можна сформувати ще цю задачу, криптопримітиву, що буде задавати безпеку нашої криптосистеми, варто поставитися максимально серйозно. По-перше, ми повинні відібрати криптопримітив, що буде задовольняти нашим функціям, які ми хочемо відповідно отримати в результаті. По-друге, цей криптопримітив повинен відповідати сучасним стандартам безпеки. І останнє – врахувати можливі уразливості і чи будуть вони критичні для вас і ваших користувачів.

2) **Керування ключами**. Ще одна важлива властивість. Ключі, це є головний рушійний механізм будь-якої криптосистеми. Неправильні ключі

зроблять нашу криптосистему неоднозначною, а витік – покладе кінець безпеці. Тому цьому пункту варто приділити багато уваги. Ключі повинні зберігатися правильно і безпечно, щоб не було можливостей незаконного проникнення і втрати ключів. Ключі також повинні ефективно і швидко генеруватися, мати правильний розподіл та підтримувати функцію оновлення

3) **Ефективність**. Система втрачає сенс, якщо вона буде неефективною. Варто розуміти ваші потужності, які ви можете забезпечити для оброблення даних. Малопроцесорні комп'ютери та смартфони не мають великого обсягу оперативної пам'яті, а отже варто подумати про дві речі, оптимальну довжину сформованих даних та ефективні методи обчислення.

4) **Інтеграція в систему**. Якщо криптосистема не інтегрована в систему, то вона працюватиме погано. Це логічно, оскільки ми потребуватимемо більше системних викликів, а отже втрачатимемо час. Гарний приклад, що ми сформували ключі викликом, але для їх отримання повинні зробити ще один виклик в систему, або користувачу необхідно встановлювати надбудови для коректної роботи криптосистеми. Якщо ваша криптосистема матиме гарний інтерфейс і підтримуватиме сумісну роботу з більшістю операційних систем, то надалі вона буде успішно матиме попит.

5) **Моніторинг**. Важливо забезпечити контроль над всіма операціями. Звичайно, зберігати їх усі за обмеженого обсягу пам'яті просто неможливо, але нехай це буде деякий журнал викликів, що дозволить простежити підозрілі виклики чи неправильні. Це перш за все необхідно для попередження можливих атак.

6) **Стійкість до атак**. Ми не можемо захиститися від усіх можливих атак. Сказати точно, в яке середовище поставить зловмисник криптосистему неможливо. Тому, варто забезпечити найпріоритетнішими властивостями захисту. Як ми знаємо, система повинна мати семантичну стійкість, нерозрізнюваність на стійкість до перетворень.

7) **Оновлення та підтримка**. Окрім вищезгаданих технічних властивостей, ми повинні забезпечити елементарні зовнішні функції.

Оновлення дозволять виправляти можливі уразливості, що були знайдені лише під час роботи. Підтримка забезпечить швидке виправлення помилок під час непередбачуваних ситуацій.

8) **Навчання та специфікації**. Написана криптосистема повинна мати чіткі інструкції з використання, зрозумілі та швидкі для сприйняття майбутніми працівниками, що обслуговуватимуть її. Тому написання специфікації з детальним описом усіх функцій є другою за складністю задачею при побудові криптосистеми, після її реалізації.

У підсумку хочемо сказати, що розробка криптосистеми, вибору алгоритму (криптопримітиву), забезпечення стійкості й визначення "стовпів" нашої криптосистеми це досить важка і відповідальна робота. Як і криптографія, що намагається віднайти найбільш стійкі та безпечні алгоритми захищення даних, криптоаналіз теж не стоїть на місці. Для побудови системи необхідно підходити комплексно та розуміти стійкість кожного кроку. Навіть генерування ключів є важливою частиною і не може бути опущена. Багато речей також залежить від задачі, складність якої є ядром вашої системи.

Але не варто забувати, яку саме задачу ми ставимо перед собою. Захист даних є доволі широким поняттям і кожна задача є індивідуальною.

## 1.2 Що таке токен

Криптографічний токен, криптотокен, USB-ключ, апаратний токен, USB-токен, електронний ключ, носій ключа електронного підпису - це все назви пристрою для зберігання криптографічних ключів та безпечного виконання операцій з ними, наприклад, підписування файлів-документів, аутентифікації в системах, шифрування та дешифрування даних. Як правило, криптотокен підключається до комп'ютера через USB, але бувають також пристрої у формфакторі смарткарт або навіть бездротові з NFC.

По суті криптотокен є мініатюрним комп'ютером з дуже обмеженим набором специфічних (як правило криптографічних) операцій. У наших

реаліях токени використовуються для зберігання закритого ключа для формування цифрового підпису та дуже активно застосовуються при спілкуванні з державними інформаційними системами, наприклад, податковим чи пенсійним фондом. В ідеальній ситуації закритий ключ ніколи не залишає токен і не існує стандартного способу його звідти витягти, тому генерацію цифрового підпису пристрій робить самостійно, використовуючи свій процесор та вбудовані криптоалгоритми. Використання криптотокенов вважається найбезпечнішим способом виконання криптографічних операцій. Також бувають «тупі» токени-контейнери, які не вміють самостійно виконувати криптографічні операції, вони по суті є просунутими «флешками» з пропрієтарним інтерфейсом, даними на яких керує криптопровайдер.

Майже всі токени використовують стандартизовані інтерфейси, які є в сучасних операційних системах, однак можуть знадобитися сторонні бібліотеки або драйвери. Як правило, у кожного виробника токенів є свої специфічні програмні інструменти для управління пристроями: ініціалізації, генерації, завантаження або видалення ключів.

## 1.3 PKS11 та його інтерфейс

Ми будемо розглядати токени, що підтримують PKCS11 — стандарт, який визначає платформонезалежний програмний інтерфейс Cryptoki до токенів та інших криптографічних пристроїв. PKCS11 входить у велике сімейство стандартів PKCS (Public-Key Cryptography Standards).

Cryptoki ізолює конкретну реалізацію всередині кожного пристрою та надає стандартний уніфікований набір функцій для виконання конкретних операцій, наприклад імпорт закритого ключа на пристрій або підписування файлу. PKCS11 реалізований практично у всіх сучасних операційних системах. У якихось він поставляється прямо з коробки (windows, macos), а в якихось вимагає установки додаткових пакетів (linux). Для окремих пристроїв можуть також знадобитися сторонні бінарні модулі-бібліотеки, що

реалізують Cryptoki, які зазвичай надаються виробником пристрою. За замовчуванням в утиліті pkcs11-tool використовується модуль opensc-pkcs11.so, який підтримує досить велику кількість пристроїв.

Токени підключаються через програмний інтерфейс роботи зі смарткартами, тобто для системи підключений USB-токен виглядає як зчитувач із вставленою смарткарткою. Щоб система розпізнала його, необхідний спеціальний драйвер, який називається IFD handler, штатно система підтримує деякі типи пристроїв, для інших потрібно встановлювати драйвери окремо.

Логічний пристрій термінології PKCS11 називається слотом (slot). Логічний пристрій, що реалізує методи Cryptoki, називається токеном (token). В одному фізичному пристрої (наприклад, USB-ключі або смарткарті) може бути кілька токенів, в цьому випадку вони займають різні слоти.

Елементи даних усередині токена (наприклад, ключі або сертифікати) називаються об'єктами (objects), для доступу до деяких з них необхідна попередня ініціалізація сесії (session) за допомогою процедури автентифікації, яку далі для стислості будемо називати «логіном». Усього в opensc визначено п'ять типів об'єктів: cert, privkey, pubkey, secrkey та data.

Кожен криптографічний алгоритм, що підтримується токеном, називається механізмом (mechanism). Пристрій може повертати список механізмів, які він підтримує.

У стандарті визначено три типи сесій:

• звичайна користувальницька, якою виконуються основні операції токена: підписування файлів, імпорт ключа та інші криптографічні функції;

• сесія офіцера безпеки/адміністратора безпеки (SO), через яку відбувається управління користувачами та пінами, у цій сесії криптографічні операції заборонені;

• анонімна сесія, у цьому режимі можна отримати список незахищених об'єктів та базову інформацію про пристрій.

Для кожного типу користувачів існує окремий пароль, який

називається піном (PIN, Personal Identification Number). Без введення PIN майже жодні операції з токеном неможливі. Також існує обмеження на кількість невірних спроб введення піна, після його перевищення користувач блокується та дані на пристрої можуть бути назавжди заблоковані. Прийнято пін користувача позначати просто як PIN, а пін офіцера безпеки як SO PIN.

Формат даних зберігання інформації про користувачів покривається іншим стандартом — PKCS15. Він дозволяє задавати різні схеми автентифікації та управління користувачами та пінами. Описана вище схема PIN і SO PIN з PKCS11 є одним з найчастіших варіантів. Зустрічаються й інші, наприклад, з PIN та PUK (Personal Unblocking Key), тут PUK є аналогом SO PIN і він використовується для зміни PIN, якщо перевищено ліміт помилкових спроб введення PIN.

Ініціалізація або персоналізація (personalization) токена - це процедура «форматування» пристрою зі знищенням всіх наявних даних та створенням структури користувачів, стандартних пінів, опціональних ключів, сертифікатів і так далі. Як правило, після персоналізації користувач токена повинен змінити початковий PIN на власний. Якщо ви забули пін або перевищено кількість спроб введення (включно з адмінським), то форматування є єдиним виходом, якщо ви збираєтеся користуватися ключем далі. Усі дані при цьому знищуються.

Найчастіше для персоналізації необхідно використовувати програмне забезпечення виробника токена.

Токен як і будь-яка смарткарта працює відповідно до стандарту ISO 7816, операції PKCS11 реалізуються через команди ISO 7816, проте деякі смарткартки також надають доступ до «файлового» інтерфейсу ISO 7816, за таким принципом працюють «тупі» токени. По суті, такі токени можна використовувати тільки в поєднанні з відповідним криптопровайдером, наприклад, КриптоПро CSP, Сигнал-КОМ CSP, ViPNet CSP. Криптопровайдер зберігає на токені дані (закриті ключі, сертифікати) у власному форматі, при цьому не використовуються жодні вбудовані в токен

механізми захисту даних.

Структура PKCS11-об'єктів у токені не є чітко формалізованою, там немає аналога файлової системи з ідентифікаторами чи іменами. Ви можете, наприклад, створити кілька закритих ключів з однаковими мітками або ідентифікаторами, або навіть без них. Однак деякі виробники накладають обмеження на PKCS11-протокол комунікації і дозволяють виконувати набір або обмежених операцій, або тільки з заданими ідентифікаторами/мітками; так робить, наприклад, Yubiko з токенами yubikey.

### 1.4 Аладдін JaCarta-2 PKI/ГОСТ

Цей пристрій підтримує найбільший набір алгоритмів: ГОСТ, RSA, EC, AES та інші. У ньому реалізовано одразу два токени: один підтримує вітчизняні ГОСТ-алгоритми, другий – закордонні (RSA).

Токен JaCarta у штатній конфігурації не працює і вимагає сторонніх бібліотек, які можна завантажити з офіційного сайту. Завантажуємо архів $jacartauc\_2.13.11.3194\_deb\_x64.zip$, вилучаємо з нього пакет $jcpkcs11 - 2\_2.7.4.540\_x64.deb$ та встановлюємо. Для роботи необхідно вказати бібліотеку libjcPKCS11-2.so.

Цей пристрій має два слоти. У першому слоті (Slot 0) знаходиться токен GOST, який вміє працювати із ключами ГОСТ. У другому слоті (Slot 1) знаходиться токен PKI (Public key infrastructure – інфраструктура на основі відкритих ключів), який вміє працювати з ключами RSA та EC.

Недоліком є те, що ГОСТ-токен у першому слоті JaCarta вимагає спеціального програмного забезпечення для адміністрування, яке у відкритому доступі відсутнє. Не рекомендується використовувати цей токен, оскільки він дуже зав'язаний на власне ПЗ і багато сценаріїв до нього непридатні.

## 1.5 Аладдін JaCarta-2 PKI/ГОСТ

Аладдін eToken PRO 72k підтримує RSA, AES, 3DES.

OpenSC із модулем за замовченням не вміє працювати з пристроєм Aladdin eToken PRO 72k (SafeNet eToken 5100), але модуль jcPKCS11-2 з минулого сценарію підтримує цей пристрій.

Бібліотека для роботи з чіпом цього токена також входить до складу SafeNet Authentication Client, який можна завантажити з офіційного сайту. У завантаженому архіві нам потрібний пакет $safenetauthenticationclient - core\_10.7.77\_amd64.deb$, встановлюємо його, інші пакети не потрібні. Для MacOS ставимо пакет SafeNet Authentication Client 10.2.pkg (він надає графічний інтерфейс для управління токеном).

Після підвантаження модуля все починає працювати в Linux.

## 1.6 Yubikey та Yubikey 4 nano

У першу чергу, Yubikey є PIV (Personal Identity Verification) пристроєм, та був розроблений за стандартами NIST SP 800-73: Interfaces for Personal Identity Verification. PKCS11 є одним з інтерфейсів, обмежений особливостями PIV такими як: неможливість створення ключів або сертифікатів з довільним ідентифікатором і т.д. І Yubikey, і Yubikey 4 nano не мають підтримки симетричних алгоритмів шифрування, але натомість підтримують базові асиметричні алгоритми.

Загалом, OpenSC частково підтримує Yubikey 4 без будь-яких додаткових бібліотек та драйверів у MacOS та Linux, але для повноцінного використання потрібно встановити додатковий модуль, який входить у пакет yubico-piv-tool. Також для деяких операцій потрібно встановити YubiKey Manager.

Варто зазначити, що для Yubikey не можна вказувати довільні ідентифікатори, тому що для ключів різних типів зарезервовані конкретні

значення, які потрібно використовувати. Крім того, для генерації та запису ключів потрібно мати права оператора безпеки та відповідний SO PIN, який у термінології Yubikey називається PIV management key.

### 1.7 Записування ключів

**Асиметричні криптосистеми** Для запису секретного ключа асиметричної криптосистеми (RSA, EC) можна згенерувати секретний ключ за допомогою openssl та імпортувати в токен спеціальною командою. Ця операція завжди вимагає логіна. Секретний ключ можна записати тільки на пристрій, який їх підтримує саме як секретні ключі.

Під час запису можна додатково вказати аргументи –id і –label, щоб задати відповідні атрибути об'єкта, за якими його можна відфільтрувати серед інших. Секретні ключі завжди записуються в захищеному режимі й при виведенні списку об'єктів без логіна не відображаються. Також зазвичай секретні ключі після запису не можна витягти з пристрою.

Відкритий ключ асиметричної криптосистеми можна отримати аналогічно з секретного. У полі Usage вказані атрибути, що описують доступні функції для цього об'єкта, наприклад для RSA це шифрування (encrypt) і перевірка цифрового підпису (verify). Як ідентифікатор вказується те саме значення, що було вказано для вихідного секретного ключа.

Відкритий ключ можна записати, тільки якщо токен підтримує відповідний механізм, а також можна прочитати з токена, причому без використання логіна.

**Симетричні криптосистеми** Для запису секретного ключа симетричної криптосистеми (AES) можна згенерувати секретний ключ за допомогою openssl, генератором випадкових чисел та імпортувати на програмний токен SoftHSM спеціальною командою.

Для успішного виконання команди пристрій має підтримувати потрібний механізм, наприклад AES підтримують eToken PRO 72k, JaCarta

і SoftHSM. Секретний ключ не можна експортувати з токена (тобто прочитати з аргументом –read-object). Деякі пристрої (наприклад, eToken PRO 72k) не підтримують запис секретного ключа, його можна тільки згенерувати на самому пристрої.

### 1.8 Перегляд доступних механізмів токена

Для отримання списку підтримуваних алгоритмів використовується аргумент –list-mechanisms або -M, причому логін не потрібен.

Кожен рядок визначає один механізм і складається з його назви та властивостей (функцій і прапорів) через кому. Якщо ідентифікатор незнайомий, то він відображається як mechtype-0xD4321033.

Основні можливі властивості для версії OpenSC 0.22.:

1) keySize=ulMinKeySize,ulMaxKeySize – визначає мінімальну і максимальну довжини підтримуваних ключів

2) digest – механізм може використовуватися для створення дайджесту

3) sign – механізм може використовуватися для створення електронного цифрового підпису

4) verify – механізм може використовуватися для перевірки електронного цифрового підпису

5) encrypt – механізм може використовуватися для шифрування

6) decrypt – механізм може використовуватися для дешифрування

7) hw – механізм працює на пристрої (hardware)

8) generate – механізм може використовуватися для генерації об'єкта

9) *generate_key_pair* – механізм може використовуватися для генерації пари відкритого та секретного ключів

10) derive – механізм може породжувати ключ з іншого ключа

11) wrap – механізм може маскувати/шифрувати (wrap) ключ для експорту

12) unwrap – механізм може імпортувати маскований/зашифрований ключ

13) $ECF_P$ – механізм можна використовувати з параметрами EC над полем $F_p$

14) $ECF_{2M}$ – механізм можна використовувати з параметрами EC над полем $F_{2m}$

15) EC parameters – механізм може використовувати параметри EC безпосередньо

16) EC OID – механізм може використовувати параметри EC через зазначення OID кривої

17) EC uncompressed – механізм може використовувати нестиснуту (uncompressed) точку еліптичної кривої

18) EC compressed – механізм може використовувати стиснуту (compressed) точку еліптичної кривої

19) EC curve name – механізм може використовувати параметри EC через вказівку кривої (curveName)

# ВИСНОВКИ

У даній роботі ми дослідили можливість реалізації криптографічних систем за допомогою інтерфейсу PKCS11. Важливим моментом є те, що кожна конкретна мета використання потребує своїх особливостей реалізації.Криптосистема повинна враховувати багато технічних нюансів, які допоможуть їй працювати коректно. Це стосується як і процесу зберігання ключів, процесу обробки даних, так і підтримка оновлення.

В контексті використання токенів, розробнику необхідно створити деяку програмну реалізацію, що зможе правильно вписати необхідні йому дані. І саме PKCS11 нам може допомогти з цим, оскільки у токені все стандартизоване і залишається лише створити правильний зв'язок.