НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ

МЕТОДИ РЕАЛІЗАЦІЇ КРИПТОГРАФІЧНИХ МЕХАНІЗМІВ

Комп'ютерний практикум № 4

Дослідження особливостей реалізації існуючих програмних систем, які використовують криптографічні механізми захисту інформації

Підгрупа 1В.

Виконали:
Сковрон Роман ФІ-42мн
Волинець Сергій ФІ-42мн
Радомир Беш ФІ-42мн

Київ — 2025

1 Мета

Отримання практичних навичок побудови гібридних криптосистем.

2 Завдання на лабораторну роботу

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.

3 Дослідження

Програмна реалізація криптосистем зумовлює декілька важливих проблем, які потребують вирішення для забезпечення ефективної та безпечної роботи системи. Ось основні з них:

Контроль доступу до ресурсів. Важливо, щоб лише авторизовані користувачі або програми мали доступ до ключів і могли виконувати криптографічні операції. Це особливо важливо при зберіганні криптографічних ключів у оперативній пам'яті ЕОМ, адже це вимагає додаткових заходів для їх захисту, оскільки оперативна пам'ять є легкодоступною для атакуючих процесів або зловмисників, якщо немає належного захисту. Одним з механізмів захисту є шифрування ключів в оперативній пам'яті є шифрування ключів. Наприклад, можна шифрувати самі ключі при їх завантаженні в оперативну пам'ять, а дешифрувати їх тільки під час використання. Також можна використовувати технології захисту на рівні апаратних модулів, таких як апаратний безпековий модуль (Hardware Security Module або HSM). HSM працює як захищений пристрій, який здійснює криптографічні операції в ізольованому середовищі, щоб забезпечити високий рівень безпеки для криптографічних ключів. Всі чутливі операції, такі як генерація, зберігання, шифрування, розшифрування та підписування даних, виконуються всередині HSM, що запобігає їх компрометації навіть у разі фізичного або логічного доступу до системи. Пристрій має захисні механізми проти фізичних втручань, таких як спроби доступу до апаратних компонентів, а також використовує складні методи контролю доступу для обмеження доступу до криптографічних ключів лише авторизованим користувачам чи процесам. HSM часто використовуються в критичних інфраструктурах, де важливо гарантувати конфіденційність і цілісність даних, наприклад, в банківських установах, урядових організаціях чи охороні здоров'я. Іншим методом є розділення ключів і їх частин. Розділені частини можна зберігати в різних областях пам'яті, що ускладнить відновлення ключа навіть за умови доступу до частини даних. На рівні операційної системи можуть використовуються механізми захисту пам'яті специфічні для певної ОС. Це, наприклад, контроль доступу на основі ролей (RBAC), яке дозволяє обмежити доступ до певних зон пам'яті. Також використовуються функції операційної

системи для обмеження доступу до адрес в пам'яті через доступні API, що дозволяє ефективно контролювати, хто і які дані може читати або змінювати.

Забезпечення стійкості до атак. Забезпечення стійкості до атак є критично важливим аспектом для програм криптографічної обробки інформації. Програми повинні бути захищені від різноманітних атак, таких як атаки на основі затримок (наприклад, атаки через аналіз часу виконання криптографічних операцій), де зловмисники можуть отримати інформацію про ключі або інші чутливі дані, аналізуючи час, необхідний для виконання певних операцій. Також важливими є захист від атаки на основі аналізу трафіку, де зловмисники можуть спробувати розшифрувати або маніпулювати зашифрованим трафіком, вивчаючи патерни або статистику передачі даних. Крім того, необхідно враховувати можливість кешування ключів або інших конфіденційних даних, що може призвести до їх несанкціонованого витоку. Наприклад, збереження ключів у кеш-пам'яті може бути вразливим до атак через побічні канали або вплив на кешування, коли атакуючий може витягнути інформацію з пам'яті, не маючи безпосереднього доступу до ключа.

Верифікація правильності роботи програми. Перевірка того, чи виконується криптографічна операція коректно, має велике значення. Важливо впевнитися, що система правильно обробляє дані та не допускає помилок у виконанні шифрування чи дешифрування. Для цього важливо проводити ретельну перевірку програмного коду та його функціональності. Це включає в себе тестування алгоритмів шифрування, дешифрування та інших криптографічних процедур на різних наборах вхідних даних, щоб переконатися, що програма дає правильні та очікувані результати. Крім того, необхідно перевірити надійність механізмів обробки ключів та автентифікації, щоб гарантувати їх правильну роботу в усіх сценаріях використання. Важливими етапами верифікації є юніт-тести для окремих модулів програми, а також інтеграційні тести, які забезпечують перевірку взаємодії між різними компонентами системи. Окрім традиційного тестування, для верифікації можна використовувати формальні методи перевірки, які дозволяють математично довести правильність виконання криптографічних алгоритмів. Це особливо важливо для систем, що працюють із чутливою інформацією. Формальні перевірки дають змогу виявити потенційні вразливості в алгоритмах або їх реалізації, які можуть призвести до помилок в шифруванні або дешифруванні. Крім того, для моніторингу правильності функціонування програми важливо впроваджувати логування всіх криптографічних операцій та їх результатів, що дозволяє швидко виявити аномалії та помилки у роботі системи, а також проводити аудит криптографічних процесів для забезпечення додаткової безпеки.

4 Інтерфейси Crypto API та PKCS 11

Crypto API (CAPI) призначений для програмної реалізації криптографії в операційних системах Windows. Він забезпечує базовий захист для ключів через програмне шифрування та використання сховищ Windows, але не підтримує апаратний захист, що може робити систему вразливою до атак через побічні канали. CAPI зручний для малих та середніх систем, однак має обмеження щодо масштабованості та безпеки на великих розподілених платформах. У CAPI ключі зберігаються в схо-

вищах Windows, доступ до яких обмежується правами користувачів. Однак захист цих ключів на рівні пам'яті не має спеціалізованих апаратних механізмів, що може становити ризик у разі компрометації програмного забезпечення або апаратних вразливостей. Відсутність апаратних засобів захисту значно знижує рівень безпеки в порівнянні з іншими рішеннями.

PKCS 11 є стандартом для інтеграції криптографії з апаратними пристроями, такими як HSM або смарт-карти. Він дозволяє зберігати ключі в апаратних пристроях, що значно підвищує рівень безпеки, оскільки всі операції з ключами здійснюються в межах цих пристроїв, знижуючи ризик витоку або компрометації даних. Це робить PKCS 11 гарним вибором для складних систем з високими вимогами до безпеки, таких як банківські або урядові рішення. PKCS 11 також підтримує гнучку інтеграцію з різними апаратними засобами, що дозволяє використовувати різні типи HSM і смарт-карт в одній системі. Високий рівень захисту від атак через бічні канали і фізичних атак, а також можливість сертифікації пристроїв (наприклад, FIPS 140-2) забезпечують надійний захист криптографічних ключів у системах, де критично важливе забезпечення безпеки даних.

Отже, Crypto API більше орієнтований на програмну реалізацію криптографії в середовищі Windows, зручний для середніх і малих за розмірами систем, де не потрібен високий рівень захисту ключів на апаратному рівні. А PKCS 11, більш універсальний і надійний стандарт для криптографії в середовищах з високими вимогами безпеки, оскільки підтримує використання апаратних засобів для захисту ключів і є стандартизованим для роботи з різними пристроями, такими як HSM і смарт-карти. Вибір між Crypto API і PKCS 11 залежить від конкретних вимог до безпеки, інтеграції з іншими системами та рівня захисту, необхідного для зберігання та обробки криптографічних ключів.

5 Реалізація

Ми реалізували простий кінцевий точок користувача (User Endpoint Terminal) з можливістю розрізнення користувачів. Імплементований функціонал шифрування та дешифрування блоковим шифром AES та змінною розміру секретного ключа, підтримується 128, 192 та 256. Також користувач має змогу розкрити свій секретний ключ.

Розрізнення користувачів відбувається за допомогою перевірки пароля. Під час першої взаємодії користувач повинен ввести свій пароль, а під час кожного іншого входу підтвердити його знання. Всі пароль зберігаються у хеш мапі користувач-пароль, проте пароль з точки зору безпеки не зберігається у вигляді пароля, а його хеш. Використовувалась хеш функція SHA-256, також цілком доцільно додатково використовувати сіл для покращення практичної стійкості, але цей аспект було вирішено проігнорувати.

Усі секретні ключі також зберігаються у вигляді хеш мапи користувач-ключ, проте ключ може бути назавжди втрачено після обирання нового ключа нового або того ж розміру.

Усі криптографічні примітиви які були використані є частиною вбудовних в сімейство Java, оскільки реалізація окремих частин таких як хеш функція SHA-256

мають у собі багато технічи аспектів.

6 Демонстрація

Для початку потрібно зареєструватися в системі.

```
--- Криптографічний Термінал ---
Введіть ваше ім'я користувача: user 1
Перше введення пароля для користувача user 1
Введіть новий пароль: 1111
Повторіть пароль: 1111
Пароль встановлено успішно.
Ключ для AES 128 біт успішно згенеровано за замовчуванням.
```

Ввівши логін користувача і створивши пароль, система автоматично генерує ключ для AES в 128 біт. Далі система перекидає нас в основне меню.

```
--- Криптографічний Термінал ---
Вхід як: user 1
1. Шифрувати текст
2. Дешифрувати текст
3. Показати ключ шифрування (Base64)
4. Налаштування AES шифрування (розмір ключа)
5. Вийти як користувач
6. Завершити програму
Виберіть операцію:
```

За потреби, можна вибрати інший розмір ключа використовуючи відповідну опцію.

```
--- Криптографічний Термінал ---
Вхід як: user 1
1. Шифрувати текст
2. Дешифрувати текст
3. Показати ключ шифрування (Base64)
4. Налаштування AES шифрування (розмір ключа)
5. Вийти як користувач
6. Завершити програму
Виберіть операцію: 4
Вибір розміру ключа для AES:
1. 128 біт
2. 192 біт
3. 256 біт
Виберіть розмір ключа: 3
Ключ для AES 256 біт успішно згенеровано.
```

Щоб побачити створений ключ, можна скористатися відповідною інструкцією.

```
--- Криптографічний Термінал ---
Вхід як: user 1
1. Шифрувати текст
2. Дешифрувати текст
3. Показати ключ шифрування (Base64)
4. Налаштування AES шифрування (розмір ключа)
5. Вийти як користувач
6. Завершити програму
Виберіть операцію: 3
Ключ шифрування (Base64): YvSEE/LdVOekLE8u6kRDFjnmR+c7Lfy41/03tR+RStc=
```

Тепер перейдемо до суті. Дана програма дозволяє шифрувати дані. Наприклад, зашифруємо текст “Some text1”.

```
--- Криптографічний Термінал ---
Вхід як: user 1
1. Шифрувати текст
2. Дешифрувати текст
3. Показати ключ шифрування (Base64)
4. Налаштування AES шифрування (розмір ключа)
5. Вийти як користувач
6. Завершити програму
Виберіть операцію: 1
Введіть текст для шифрування: Some text1
Зашифрований текст: 86YtKLMcMnCyt70wPTnetQ==
```

А тепер розшифруємо його.

```
--- Криптографічний Термінал ---
Вхід як: user 1
1. Шифрувати текст
2. Дешифрувати текст
3. Показати ключ шифрування (Base64)
4. Налаштування AES шифрування (розмір ключа)
5. Вийти як користувач
6. Завершити програму
Виберіть операцію: 2
Введіть зашифрований текст для дешифрування: 86YtKLMcMnCyt70wPTnetQ==
Розшифрований текст: Some text1
```

Якщо взяти шифротекст який не є дійсним, то розшифрування не відбудеться.

```
--- Криптографічний Термінал ---
Вхід як: user 1
1. Шифрувати текст
2. Дешифрувати текст
3. Показати ключ шифрування (Base64)
```

```
4. Налаштування AES шифрування (розмір ключа)
5. Вийти як користувач
6. Завершити програму
Виберіть операцію: 2
Введіть зашифрований текст для дешифрування: aaaaaaaaaaaaaaaaaaaaaaaa
Помилка при дешифруванні: Помилка при дешифруванні:
    Input length must be multiple of 16 when decrypting with padded cipher
```

Далі, змінемо користувача, і зашифруємо текст “Some text1” знову.

```
--- Криптографічний Термінал ---
Вхід як: user 1
1. Шифрувати текст
2. Дешифрувати текст
3. Показати ключ шифрування (Base64)
4. Налаштування AES шифрування (розмір ключа)
5. Вийти як користувач
6. Завершити програму
Виберіть операцію: 5
Вихід як користувач user 1
--- Криптографічний Термінал ---
Введіть ваше ім’я користувача: user2
Перше введення пароля для користувача user2
Введіть новий пароль: 1234
Повторіть пароль: 1234
Пароль встановлено успішно.
Ключ для AES 128 біт успішно згенеровано за замовчуванням.

--- Криптографічний Термінал ---
Вхід як: user2
1. Шифрувати текст
2. Дешифрувати текст
3. Показати ключ шифрування (Base64)
4. Налаштування AES шифрування (розмір ключа)
5. Вийти як користувач
6. Завершити програму
Виберіть операцію: 4
Вибір розміру ключа для AES:
1. 128 біт
2. 192 біт
3. 256 біт
Виберіть розмір ключа: 3
Ключ для AES 256 біт успішно згенеровано.

--- Криптографічний Термінал ---
Вхід як: user2
1. Шифрувати текст
```

```
2. Дешифрувати текст
3. Показати ключ шифрування (Base64)
4. Налаштування AES шифрування (розмір ключа)
5. Вийти як користувач
6. Завершити програму
Виберіть операцію: 1
Введіть текст для шифрування: Some text1
Зашифрований текст: 0H6oS0Jec/W084Z5myc2SA==


--- Криптографічний Термінал ---
Вхід як: user2
1. Шифрувати текст
2. Дешифрувати текст
3. Показати ключ шифрування (Base64)
4. Налаштування AES шифрування (розмір ключа)
5. Вийти як користувач
6. Завершити програму
Виберіть операцію: 2
Введіть зашифрований текст для дешифрування: 0H6oS0Jec/W084Z5myc2SA==
Розшифрований текст: Some text1
```

Як видно, два користувачі зашифрували один і той самий текст по різному. Якщо спробувати розшифрувати неправильний текст, то отримаємо наступне.

```
--- Криптографічний Термінал ---
Вхід як: user2
1. Шифрувати текст
2. Дешифрувати текст
3. Показати ключ шифрування (Base64)
4. Налаштування AES шифрування (розмір ключа)
5. Вийти як користувач
6. Завершити програму
Виберіть операцію: 2
Введіть зашифрований текст для дешифрування: 86YtKLMcMnCyt70wPTnetQ==
Помилка при дешифруванні: Помилка при дешифруванні:
    Given final block not properly padded. Such issues can arise if a
    bad key is used during decryption.
```

А тепер продемонструємо що буде, якщо ввести невірний пароль при вході.

```
--- Криптографічний Термінал ---
Вхід як: user2
1. Шифрувати текст
2. Дешифрувати текст
3. Показати ключ шифрування (Base64)
4. Налаштування AES шифрування (розмір ключа)
```

```
5. Вийти як користувач
6. Завершити програму
Виберіть операцію: 5
Вихід як користувач user2
--- Криптографічний Термінал ---
Введіть ваше ім'я користувача: user 1
Введіть пароль: 1112
Невірний пароль! Спробуйте знову.
Введіть пароль: 1234
Невірний пароль! Спробуйте знову.
Введіть пароль: 1111
Вхід успішний.
```

7 Висновки

У процесі реалізації криптосистеми були розглянуті важливі аспекти безпеки, зокрема захист ключів, контроль доступу до чутливої інформації та перевірка правильності роботи програми. Для захисту криптографічних ключів у оперативній пам'яті були запропоновані різні методи, зокрема шифрування ключів, використання апаратних модулів безпеки (HSM), а також методи розподілу та захисту даних в операційних системах.

Забезпечення стійкості системи до атак, таких як атаки на основі аналізу часу чи трафіку, потребує додаткових заходів, як-то захист від аналізу часу або використання спеціальних протоколів для захисту даних при передачі. Верифікація правильності функціонування програми є важливою для гарантії коректного виконання криптографічних операцій, тому доцільно використовувати формальні методи перевірки і тестування.

Що стосується порівняння інтерфейсів Crypto API та PKCS 11, то кожен з них має свої переваги. Crypto API забезпечує базову програмну підтримку криптографії для операційних систем Windows, але має обмеження з точки зору безпеки. PKCS 11, в свою чергу, підтримує інтеграцію з апаратними пристроями, що дозволяє значно підвищити рівень захисту, і тому є кращим вибором для складних систем з високими вимогами до безпеки.

Загалом, під час реалізації криптосистеми були враховані основні аспекти безпеки, необхідні для захисту даних та забезпечення правильної роботи програми, що дозволяє створювати безпечніші та більш надійні системи для обробки чутливої інформації.