НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ СІКОРСЬКОГО»

ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ

Лабораторна робота 4

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

Підгрупа 2А

Виконали:

Галіца О.О.

Паршин О.Ю.

Литвиненко Ю.С.

ФІ-22мн

Перевірила:

Байденко П.В.

Київ 2022

1 CryptoAPI

1.1 Основні відомості

CryptoAPI – інтерфейс програмування додатків, який забезпечує розробників стандартним набором функцій для роботи з криптопровайдером. Входить до складу операційних систем Microsoft. Більшість функцій CryptoAPI підтримуються, починаючи з Windows 2000.

Модна розділити весь інтерфейс Crypto API на 5 функціональних груп:

1. Базові криптографічні функції:
   - функції шифрування/розшифрування даних;
   - функції хешування та отримання цифрового підпису даних;
   - функції ініціалізації криптопровайдера та роботи з отриманим контекстом;
   - функції генерації ключів;
   - функції обміну ключами.
2. Функції кодування/декодування. Під кодуванням в даному випадку мається на увазі отримання інформації, закодованої у форматі ASN.1.
3. Функції роботи із сертифікатами.
4. Високорівневі функції обробки криптографічних повідомлень.
5. Низькорівневі функції обробки криптографічних повідомлень.

Криптопровайдером називають незалежний модуль, що забезпечує безпосередню роботу з криптографічними алгоритмами. Кожен криптопровайдер повинен забезпечувати:

1. реалізацію стандартного інтерфейсу криптопровайдера;
2. роботу з ключами шифрування, призначеними для забезпечення роботи алгоритмів, специфічних для цього криптопровайдера;
3. неможливість втручання третіх осіб у схему роботи алгоритмів.

Криптопровайдери реалізуються у вигляді бібліотек, що динамічно завантажуються (DLL). Таким чином, досить важко вплинути на хід алгоритму, реалізованого в криптопровайдері, оскільки всі компоненти криптосистеми Windows повинні мати цифровий підпис, а отже також підписується і DLL криптопровайдера. Не повинно біти можливості змінити алгоритм у криптопровайдері шляхом зміни параметрів цього алгоритму. Таким чином вирішується задача забезпечення цілісності алгоритмів криптопровайдера.

Завдання забезпечення цілісності ключів шифрування вирішується з використанням контейнера ключів. Контейнером ключів називають частину бази даних ключів, яка містить пару ключів для обміну ключами та формування цифрового підпису. Як сховища пар ключів використовують, наприклад, область тимчасової пам'яті, ділянку реєстру, файл на диску, смарт-картки.

1.2 Вразливості

За останній час було виявлено всього дві вразливості CryptoAPI. Спочатку розглянемо криптографічний баг, який є небезпечним для Windows 10, Windows Server 2019 і Windows Server 2016. Уразливість у криптографічній бібліотеці crypt32.dll у Windows, яку назвали **CVE-2020-0601**, полягає в тому, що Windows CryptoAPI (Crypt32.dll) не повністю перевіряє сертифікати ECC (elliptic curve cryptography). Вразливі версії Windows перевіряють три параметри ECC, пропускаючи при цьому четвертий, критичний.

Зловмисник може використовувати цю вразливість, щоб підписати шкідливу програму так, щоб система прийняла файл за легітимний. Найгірше те, що вразливість також може використовуватися для підробки цифрових сертифікатів, що використовуються для шифрованих комунікацій. Фактично успішна експлуатація проблеми дозволяє проводити MitM-атаки і розшифровувати конфіденційну інформацію про підключення користувача.

Ще одну слабкість було виявлено у жовтні 2022 року. Ця вразливість є дуже серйозною та відома як **CVE-2022-34689**. Вона виникла у результаті неправильно реалізованих схем автентифікації, які піддаються атакам спуфінгу. Технічні деталі невідомі, а експлойт недоступний.

Про цю вразливість доволі мало відомостей. Зловмисник може маніпулювати наявним загальнодоступним сертифікатом x.509, щоб підробити свою особу та виконати такі дії, як, наприклад, автентифікація. Також відомо, що відбувається повна втрата цілісності. Наприклад, зловмисник може змінити будь-який (або усі) файли, захищені зараженим компонентом. Можна змінити лише деякі файли, але зловмисна зміна призведе до прямих серйозних наслідків. Зловмисник не потребує доступу до налаштувань або файлів для здійснення атаки. Спеціальних умов доступу або пом'якшувальних обставин також не існує. Зловмисник може повторно провести атаку на вразливий компонент.

1.3 Недоліки CryptoAPI

На жаль, CryptoAPI вже морально застарів, і хоча вже досить давно вийшов CryptoAPI 2.0 для .NET архітектури – це лише для C#, для C/C++ альтернативи немає, і навряд чи буде. Застарілість заключається у відсутності підтримки нових можливостей мов, що перетворюється у функції, які приймають купу параметрів не зовсім зрозуміло для чого, наприклад, ось сигнатура для деякого уявного методу шифрування:

```
BOOL CryptEncrypt(
    HCRYPTKEY hKey,
    HCRYPTHAS hHash,
    BOOL Final,
    DWORD dwFlags,
    BYTE* pbData,
    DWORD* pdwDataLen,
    DWORD dwBufLen
);
```

```
BOOL CryptHashData(HCRYPTHASH hHash,
    BYTE* pbData,
    DWORD dwDataLen,
    DWORD dwFlags
);

BOOL CryptSignHash(HCRYPTHASH hHash,
    DWORD dwKeySpec,
    LPCTSTR sDescription,
    DWORD dwFlags,
    BYTE* pbSignature,
    DWORD* pdwSigLen
);
```

Багато параметрів для якихось довжин, для додаткових параметрів, які, зрозуміло, передбачалось для ручної роботи з пам'яттю за допомогою вказівників, але в C++ давно є багато вбудованих реалізованих структур даних та в цілому оновлень, які дозволяють працювати з усім цим більш безболісно і зрозуміло (наприклад, `std::unique_ptr`, `std::map`, `std::vector`, `std::pair` тощо).

Тому в даній лабораторній роботі інтерфейси були підігнані під веб-стандарт Web Crypto API.

1.4 Web Cryptography API

З назви зрозуміло, що це Web стандарт, причому для реалізації в JavaScript. В основі всього лижить деякий об'єкт CryptoSubtle, через який відбувається взаємодія з криптографічними функціями. Наведемо його інтерфейс:

```
interface SubtleCrypto {
    Promise<any> encrypt(
        AlgorithmIdentifier algorithm,
        CryptoKey key,
        BufferSource data
    );

    Promise<any> decrypt(
        AlgorithmIdentifier algorithm,
        CryptoKey key,
        BufferSource data
    );

    Promise<any> sign(
        AlgorithmIdentifier algorithm,
```

```
            CryptoKey key,
            BufferSource data
        );

        Promise<any> verify(
            AlgorithmIdentifier algorithm,
            CryptoKey key,
            BufferSource signature,
            BufferSource data
        );

        Promise<any> digest(
            AlgorithmIdentifier algorithm,
            BufferSource data
        );

        Promise<any> generateKey(
            AlgorithmIdentifier algorithm,
            boolean extractable,
            sequence<KeyUsage> keyUsages
        );
    }
```

Одразу видно, наскільки сигнатури стали чистішими і зрозумілішими, у функції передаються лише ті параметри, які чисто інтуїтивно і очікуєш побачити. І хоча це стандарт для JavaScript – можливості сучасного C++ дозволяють реалізувати подбіний інтерфейс без значних труднощів, що і було зроблено в рамках цієї лабораторної роботи.

2 Результати

В ході лабораторної роботи була реалізована криптосистема Ель-Гамаля з параметризованою довжиною ключа. В якості допоміжної бібліотеки для роботи з довгою арифметикою була використана бібліотека OpenSSL.

Тестове повідомлення: D4D2110984907B5625309D956521BAB4157B8B1ECE04043249A3D379AC112E5B9AF44E721E148D88A942744CF56A06B92D28A0DB950FE4CED2B41A0BD38BCE7D0BE1055CF5DE38F2A588C2C9A79A75011058C320A7B661C6CE1C36C7D870758307E5D2CF07D9B6E8D529779B6B2910DD17B6766A7EFEE215A98CAC300F2827DB

Рис. 1: Шифрування та розшифрування

Рис. 2: Підписування та перевірка підпису