НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені Ігоря СІКОРСЬКОГО»
Навчально-науковий фізико-технічний інститут
Кафедра математичних методів захисту інформації

Звіт до лабораторної №4

за темою:
Особливості існуючих прикладних програм, що використовують криптографічні механізми для захисту інформації

Оформлення звіту:
Юрчук Олексій, ФІ-52мн

22 жовтня 2025 р.
м. Київ

ЗМІСТ

1 Introduction

Інформаційна безпека стала критично важливим питанням у сучасних обчислювальних системах, де передача та зберігання конфіденційних даних вимагають надійних механізмів захисту. Криптографічні системи є основою інформаційної безпеки, забезпечуючи конфіденційність, цілісність, автентифікацію та незаперечність дій [1].

В лабораторній роботі я розглянув особливості існуючих програмних систем, що реалізують криптографічні механізми, з особливим акцентом на асиметричній криптографії, стандартизованих інтерфейсах та оцінці вразливості. Еволюція криптографічного програмного забезпечення призвела до розробки стандартизованих інтерфейсів прикладного програмування (API), таких як CryptoAPI від Microsoft та стандарти криптографії з відкритим ключем (PKCS), розроблені RSA Laboratories. Ці стандарти сприяють взаємодії між різними криптографічними реалізації та забезпечення послідовних властивостей безпеки на різних платформах [2, 3].

Ця теоретична основа створює контекст для практичної реалізації криптографічних систем, зокрема і криптосистеми ElGamal з використанням бібліотеки OpenSSL, та дослідження потенційних вразливостей у криптографічних провайдерах, які можуть виникнути через недосконаліть механізмів захисту операційної системи.

2 Fundamentals of Cryptographic Systems

2.1 Класифікація криптографічних алгоритмів

Криптографічні алгоритми широко класифікуються на дві категорії: симетрична та асиметрична криптографія. Симетричні алгоритми, такі як AES та DES, використовують один і той самий ключ для операцій шифрування та розшифрування. Хоча симетричні системи є обчислювально ефективними, вони стикаються з проблемою безпечного розподілу ключів [4].

Асиметрична криптографія або криптографія з відкритим ключем вирішує проблему розподілу ключів за допомогою математично пов'язаних пар ключів: відкритого ключа для шифрування та закритого ключа для розшифрування. Безпека асиметричних систем базується на припущені про обчислювальну складність, таких як складність розкладання великих цілих чисел на множники (RSA) або обчислення дискретних логарифмів (ElGamal, DSA) [5].

2.2 Криптосистема ElGamal

Криптосистема ElGamal, запропонована Тахером ЕльГамалем у 1985 році, базується на проблемі дискретного логарифму в скінченних полях [6]. Система працює в мультиплікативній групі цілих чисел за модулем великого простих числа p, де проблема дискретного логарифму вважається обчислювально нерозв'язною.

Генерація ключа:

1. Вибирати велике просте число p та генератор g мультиплікативної групи $\mathbb{Z}_p^*$
2. Вибирати випадковий private key x, де $1 \leq x \leq p-2$
3. Обчислити public key $y = g^x \pmod{p}$
4. Public key: (p, g, y); Private key: x

Шифрування: Для зашифрування повідомлення m, де $0 \leq m \leq p-1$ необхідно:

1. Вибрати випадковий ephemeral key k, де $1 \leq k \leq p-2$
2. Обчислити $c_1 = g^k \pmod{p}$

3. Обчислити $c_2 = m \cdot y^k \pmod{p}$
4. Шифротекст: (c_1, c_2)

Розшифрування: Щоб розшифрування шифротексту (c_1, c_2):

$$m = c_2 \cdot (c_1^x)^{-1} \pmod{p}$$

Безпека ElGamal базується на обчислювальному припущенні Діффі-Хеллмана, що робить її стійкою до відомих криптоаналітичних атак при правильній реалізації з достатньо великими параметрами [6].

2.3 Криптографічні примітиви

Сучасні криптографічні системи надають чотири основні security issues:

Confidentiality гарантує, що інформація доступна тільки уповноваженим сторонам за допомогою механізмів шифрування. Конфіденційність можуть забезпечувати як симетричні, так і асиметричні алгоритми, хоча асиметричні системи зазвичай використовуються для обміну ключами через обчислювальні накладні витрати.

Integrity гарантує, що інформація не була змінена під час передачі або зберігання. Хеш-функції та коди автентифікації повідомлень (MAC) забезпечують перевірку цілісності, не перешкоджаючи спробам модифікації.

Authentication перевіряє ідентичність сторін, що спілкуються. Цифрові підписи, засновані на асиметричній криптографії, забезпечують надійну аутентифікацію, демонструючи володіння приватним ключем, що відповідає відомому публічному ключу.

Non-repudiation запобігає спробам сторін заперечувати свої дії. Цифрові підписи забезпечують невідмовність, створюючи докази того, що конкретна сторона створила повідомлення [1].

3 Стандартизовані криптографічні інтерфейси

3.1 Microsoft CryptoAPI

Криптографічний інтерфейс прикладного програмування (CryptoAPI) від Microsoft забезпечує основу для інтеграції криптографічних функцій у Windows додатки [3]. Архітектура складається з декількох рівнів:

Application рівень: додатки взаємодіють з CryptoAPI за допомогою функцій високого рівня, які абстрагують криптографічні операції.

CryptoAPI рівень: він забезпечує стандартизовані інтерфейси для криптографічних операцій, включаючи хешування, шифрування, цифрові підписи та управління сертифікатами.

Cryptographic Service Provider (CSP) рівень: CSP реалізують фактичні криптографічні алгоритми. У системі може співіснувати декілька CSP, що дозволяє додаткам вибирати відповідних постачальників на основі вимог безпеки або відповідності нормативним вимогам.

Архітектура CryptoAPI підтримує як програмні, так і апаратні криптографічні реалізації. Hardware security modules (HSM) можуть бути інтегровані як CSP, забезпечуючи підвищений захист ключів за допомогою спеціального криптографічного обладнання.

Основні фічі в CryptoAPI:

- Незалежність від алгоритмів завдяки provider abstraction
- Підтримка криптографічних апаратних токенів
- Certificate-based public key infrastructure (PKI)
- Безпечне зберігання та управління ключами
- Генерація криптографічних випадкових чисел

3.2 Public-Key Cryptography Standards (PKCS)

Сімейство стандартів PKCS, розроблене RSA Laboratories, визначає сумісні криптографічні формати даних і протоколи [2]. Ці стандарти забезпечують сумісність між різними криптографічними реалізаціями та сприяють безпечному обміну даними між гетерогенними системами.

PKCS #1 визначає схеми шифрування та підпису RSA, включаючи методи заповнення (PKCS#1 v1.5 та OAEP), що запобігають різним криптоаналітичним атакам. Стандарт визначає формати даних для відкритих та закритих ключів RSA, зашифрованих повідомлень та цифрових підписів.

PKCS #3 описує протокол узгодження ключів Діффі-Хеллмана, що дозволяє двом сторонам встановити спільний секрет через незахищений канал. Цей стандарт визначає параметри домену та процедури походження ключів.

PKCS #5 визначає схеми шифрування на основі паролів, включаючи функції створення ключів (PBKDF2) які перетворюють паролі в криптографічні ключі з відповідною ентропією. Ці функції включають якількість ітерацій і salt values (випадкові дані додані до паролю перед шифруванням) для протидії атакам за словником.

PKCS #7 (зараз замінений на RFC 5652 Cryptographic Message Syntax) визначає формати для цифрового підпису, шифрування або автентифікації даних. Цей стандарт підтримує кілька алгоритмів підпису та шифрування, що забезпечує гнучкість політик безпеки.

PKCS #11 (Cryptoki) визначає незалежний від платформи API для криптографічних токенів, таких як смарт-карти та HSM. Цей стандарт дозволяє додаткам отримувати доступ до криптографічних служб без використання коду, специфічного для конкретного пристрою.

PKCS #12 визначає портативний формат для зберігання та передачі приватних ключів, сертифікатів та інших криптографічних даних, які зазвичай захищені шифруванням на основі пароля.

3.3 Архітектура бібліотеки OpenSSL

OpenSSL — це надійна реалізація протоколів SSL/TLS з відкритим кодом та універсальна бібліотека для реалізації криптографічних механізмів [7]. Бібліотека надає комплексні криптографічні функції, зокрема:

Symmetric Cryptography: Реалізація алгоритмів, включаючи AES, DES, 3DES, Blowfish, RC4 та ChaCha20, з підтримкою декількох режимів роботи (ECB, CBC, CFB, OFB, CTR, GCM).

Asymmetric Cryptography: Підтримка RSA, DSA, Diffie-Hellman та криптографії на еліптичних кривих (elliptic curve cryptography) (ECC). Хоча OpenSSL не включає вбудоване шифрування для ElGamal, бібліотека надає необхідні примітиви такі як модулярне піднесення до степеня, генерація простих чисел, генерація випадкових чисел для власне реалізації ElGamal.

Hash Functions: Реалізація сімейства MD5, SHA-1, SHA-2 (SHA-224, SHA-256, SHA-384, SHA-512) та SHA-3, а також кодів автентифікації повідомлень (HMAC).

Certificate Management: Генерація та аналіз сертифікатів X.509, підтримка операцій інфраструктури відкритих ключів

Random Number Generation: Криптографічно захищений генератор псевдовипадкових чисел (Cryptographically secure pseudorandom number generator – CSPRNG), необхідний для генерації ключів та nonce creation.

Архітектура OpenSSL відокремлює криптографічні примітиви від реалізації протоколів, що дозволяє розробникам використовувати низькорівневі криптографічні функції для власних додатків, таких як реалізація криптосистеми ElGamal.

4 Розробка Cryptographic Software

4.1 Вимоги до безпеки

Реалізація криптографічних систем вимагає ретельної уваги до вимог безпеки, що виходять за межі алгоритмічної коректності:

- **Управління ключами:** Безпечне генерування, зберігання, розподіл та знищення криптографічних ключів є критично важливими питаннями безпеки. Ключі повинні генеруватися за допомогою криптографічно захищених генераторів випадкових чисел з достатньою ентропією. Приватні ключі потребують захисту від несанкціонованого доступу за допомогою шифрування, контролю доступу або апаратних модулів безпеки [4].

- **Генерація випадкових чисел:** Криптографічні операції залежать від високої якості випадковості. Якщо випадкові числа є передбачуваними, то вони ставлять під загрозу безпеку, дозволяючи відновлення ключів або підробку підписів. Операційні системи надають джерела ентропії з апаратних переривань, введення користувача та шумів з навколишнього середовища, які криптографічні бібліотеки повинні належним чином збирати та обробляти.

- **Стійкість до Side-Channel attack:** Вразливості реалізації можуть призвести до витоку секретної інформації через коливання часу, споживання енергії, електромагнітне випромінювання або моделі доступу до кешу. Алгоритми з постійним часом і безпечні практики кодування зменшують ризик атак на основі часу [8].

- **Атаки спричинені помилками:** Апаратні помилки, спричинені маніпуляціями з навколишнім середовищем (перепади напруги, коливання температури, випромінювання), можуть призвести до розкриття секретних ключів криптографічних реалізацій. Надійні механізми виявлення та перевірки помилок забезпечують захист від атак через введення помилок [9].

4.2 Вибір параметрів

Криптографічна безпека критично залежить від вибору параметрів. Для реалізації ElGamal:

- **Вибір простих чисел:** просте число p повинно мати розмір не менше 2048 bits для нинішньої безпеки, а для довгострокового захисту рекомендується використовувати 3072 або 4096 bits. Просте число має бути ”safe prime” ($p = 2q + 1$ де q також просте), щоб забезпечити велику підгрупу порядком простого числа.

- **Вибір генератора:** Генератор g повинен мати великий порядок у мультиплікативній групі. Для безпечних простих чисел вибір $g = 2$ або перевірка того $g^q \pmod p \neq 1$ забезпечує відповідний порядок.

- **Quality випадкових чисел:** Private key x та ephemeral keys k повинні генеруватися за допомогою криптографічно захищених генераторів випадкових чисел. Повторне використання тимчасових ключів або використання передбачуваних значень катастрофічно порушує безпеку.

5 Вразливості в криптографічних реалізаціях

5.1 Механізми захисту в операційній системі

Криптографічне програмне забезпечення залежить від служб безпеки окремої операційної системи, включаючи захист пам’яті, контроль доступу та ізоляцію процесів. Недосконалість цих механізмів створює такі вразливі місця:

1. **Memory Disclosure:** Page file swapping може записувати криптографічні ключі на диск у вигляді звичайнісінького тексту. Дампи пам'яті, що генеруються під час збою системи або, навпаки, налагодження, можуть потенційно розкрити конфіденційні дані. Функції безпечного розподілу пам'яті запобігають обміну та забезпечують обнулення(очищення) пам'яті після використання.

2. **Side-Channel Leakage attack:** Планування операційної системи та управління спільними ресурсами створюють канали синхронізації. Атаки на синхронізацію кешу використовують спільні кеші процесора для визначення криптографічних ключів шляхом аналізу моделей доступу. Вразливості спекулятивного виконання (Spectre, Meltdown) обходять ізоляцію пам'яті, потенційно можучи розкрити криптографічні матеріали.

3. **Вразливості API:** Криптографічні API можуть містити деякі конструктивні недоліки, що завдяки яким можливі зловживання. Неправильна обробка помилок може призвести до витоку інформації про приватні ключі через повідомлення про помилки або різницю в часі між дійсними та недійсними операціями.

5.2 Атаки, залежно від реалізації

Окрім алгоритмічної безпеки, безпека криптографічних систем багато чим залежить від реалізації:

1. **Атака на синхронізацію (Timing Attacks):** Варіації часу виконання на основі секретних даних дозволяють відновити ключ. Умовні розгалуження та залежний від даних доступ до пам'яті створюють канали синхронізації. Constant time реалізації усувають залежні від даних варіації синхронізації [8].

2. **Buffer Overflow:** Порушення безпеки пам'яті дозволяють додавати код або отримувати довільний доступ до пам'яті. Криптографічні бібліотеки повинні використовувати перевірку limits і безпечно управляти пам'яттю.

3. **Атака на бічні канали:** Аналіз потужності та аналіз електромагнітного випромінювання дозволяють відновити ключі з фізичних пристроїв. Бічні канали програмного забезпечення включають синхронізацію кешу, прогнозування розгалужень та спекулятивне виконання.

4. **Введення помилок(Fault Injection):** Включення(додавання) обчислювальної помилки під час криптографічних операцій може призвести до розкриття секретного ключа. Надлишкові обчислення та виявлення помилок забезпечують захист від атак з використанням computational [9].

5.3 Оцінка стійкості криптосистеми до атак

- **Аналіз часу:** Статистичний аналіз часу виконання операцій для різних вхідних даних дозволяє ідентифікувати канали часу. Перевірка з постійним часом забезпечує незалежність часу виконання від вхідних даних.

- **Безпека пам'яті:** Регулярний аналіз виявляють переповнення буфера, вразливості використання після звільнення та інші проблеми пошкодження пам'яті.

- **Витік інформації:** Моніторинг системних журналів, повідомлень про помилки та результатів налагодження дозволяє виявити ненавмисне розкриття інформації і запобігти навмисному витоку.

6 Практична реалізація Framework-у

6.1 Реалізація ElGamal за допомогою OpenSSL

Реалізація криптосистеми ElGamal за допомогою OpenSSL вимагає використання арифметики великих чисел та засобів генерації випадкових чисел бібліотеки. Процес реалізації включає:

Parameter Generation: Використання функцій генерації простих чисел OpenSSL для створення безпечних простих чисел відповідного розміру. Функція `BN_generate_prime_ex()` генерує прості числа певного розміру і властивостями.

Key Pair Generation: Вибір відповідних генераторів і обчислення відкритих ключів за допомогою модулярного піднесення до степеня за допомогою функції `BN_mod_exp()`. A private keys генеруються за допомоги криптографічно захищеного генератора випадкових чисел.

Encryption Implementation: Генерація ephemeral keys для кожної операції шифрування та виконання модульної арифметики для обчислення компонентів шифрованого тексту. Кодування повідомлень може вимагати схем заповнення для обробки повідомлень, коротших за модуль.

Decryption Implementation: Обчислення обернених за модулем з використанням функції `BN_mod_inverse()` та виконання модульного множення для відновлення відкритого тексту.

6.2 Методологія тестування безпеки

Комплексне тестування безпеки оцінює алгоритмічну правильність і стійкість реалізації до атак:

- **Functional Testing:** Перевірка правильності шифрування та дешифрування для різних розмірів повідомлень і ключових параметрів. Тестування граничних умов і обробки помилок. **Timing Analysis:** Вимірювання часу виконання операцій для виявлення залежностей від даних. Статистичний аналіз визначає потенційні канали часу. **Тестування безпеки пам'яті:** Використання засобів очищення пам'яті (AddressSanitizer, MemorySanitizer) для виявлення помилок пам'яті. Fuzzing із випадковими вхідними даними визначає умови збою. **Privilege Testing:** Оцінка захисту ключів за різних конфігурацій доступу користувачів

7 Conclusion

Безпека криптографічного програмного забезпечення виходить за межі алгоритмічної міцності і охоплює деталі реалізації, вибір параметрів та стійкість до атак через побічні канали. Механізми захисту операційної системи відіграють вирішальну роль у забезпеченні безпеки криптографічних операцій, проте недосконаліть цих механізмів створює потенційні вразливості. Атаки на реалізацію, включаючи аналіз часу, введення помилок та розкриття пам'яті, загрожують навіть математично безпечним алгоритмам.

Практична реалізація криптосистеми ElGamal з використанням бібліотеки OpenSSL демонструє застосування теоретичних принципів у розробці криптографічного програмного забезпечення в реальному світі. Ідеальна реалізація повинна враховувати генерацію ключів, якість випадкових чисел, вибір параметрів та стійкість до атак через побічні канали, а також дотримуватися встановлених стандартів.

Систематична оцінка стійкості криптографічного провайдера до атак, що використовують недоліки механізмів захисту операційної системи, вимагає комплексного тестування, включаючи аналіз часу, перевірку безпеки пам'яті та тестування ескалації привілеїв. Цей багатогранний підхід забезпечує надійну криптографічну реалізацію, здатну захистити конфіденційну інформацію в сучасних обчислювальних середовищах.

References

[1] William Stallings. *Cryptography and Network Security: Principles and Practice*. 7th. Pearson, 2017.

[2] RSA Laboratories. *PKCS #1: RSA Cryptography Standard*. Tech. rep. Version 2.2. RFC 8017. RSA Security Inc., 2012.

[3] *CryptoAPI System Architecture*. Microsoft Developer Network. Microsoft Corporation. 2018.

[4] Niels Ferguson, Bruce Schneier, and Tadayoshi Kohno. *Cryptography Engineering: Design Principles and Practical Applications*. Wiley, 2010.

[5] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[6] Taher ElGamal. «A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms». In: *IEEE Transactions on Information Theory* 31.4 (1985), pp. 469–472.

[7] *OpenSSL Documentation*. OpenSSL Software Foundation. 2023. URL: https://www.openssl.org/docs/.

[8] Paul C. Kocher. «Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems». In: *Advances in Cryptology — CRYPTO '96*. Springer, 1996, pp. 104–113.

[9] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. «On the Importance of Checking Cryptographic Protocols for Faults». In: *Advances in Cryptology — EUROCRYPT '97* (1997), pp. 37–51.

A Скорочення

ASLR Address Space Layout Randomization - Security technique that randomizes memory addresses

ASN.1 Abstract Syntax Notation One - Standard for describing data structures

CRT Chinese Remainder Theorem - Optimization for RSA operations

DEP Data Execution Prevention - Memory protection preventing code execution from data pages

CSP Cryptographic Service Provider - CryptoAPI implementation module

DPAPI Data Protection API - Windows API for encrypting user data

HSM Hardware Security Module - Dedicated cryptographic hardware

IND-CCA2 Indistinguishability under Adaptive Chosen Ciphertext Attack

IND-CPA Indistinguishability under Chosen Plaintext Attack

KSP Key Storage Provider - CNG key storage implementation

OAEP Optimal Asymmetric Encryption Padding - RSA padding scheme

PEM Privacy Enhanced Mail - Base64 ASCII encoding format

PKCS Public-Key Cryptography Standards - RSA Security standards