BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐẠI HỌC ĐẠI NAM

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO BÀI TẬP LỚN

NHẬP MÔN AN TOÀN BẢO MẬT THÔNG TIN

GAME "HỆ THỐNG MÃ HÓA NGÂN HÀNG"

Sinh viên thực hiện : Hà Huy Khánh Dương
Hoàng Văn Dũng

Ngành : Công nghệ thông tin

Giảng viên hướng dẫn : ThS. Lê Thị Thùy Trang

Lời cảm ơn

Trong quá trình học tập và nghiên cứu tại trường Đại học Đại Nam, mỗi sinh viên đều được tạo cơ hội để vận dụng kiến thức lý thuyết vào thực tiễn thông qua các bài tập lớn, đồ án hoặc khóa luận. Đây không chỉ là yêu cầu bắt buộc của chương trình đào tạo mà còn là cơ hội quý báu giúp sinh viên rèn luyện tư duy logic, kỹ năng làm việc độc lập và khả năng giải quyết vấn đề thực tế.

Báo cáo bài tập lớn này được thực hiện với mục tiêu củng cố và áp dụng những kiến thức đã học vào một đề tài cụ thể. Trong quá trình thực hiện, nhóm chúng em đã cố gắng tìm hiểu tài liệu, phân tích, thiết kế và triển khai các nội dung liên quan đến đề tài. Mặc dù đã có nhiều nỗ lực nhưng do kiến thức và kinh nghiệm còn hạn chế, bài báo cáo chắc chắn vẫn không tránh khỏi những thiếu sót.

Chúng em xin chân thành cảm ơn ThS. Lê Thị Thùy Trang đã tận tình hướng dẫn, hỗ trợ và tạo điều kiện thuận lợi cho chúng em trong suốt quá trình thực hiện bài tập lớn này.

Rất mong nhận được sự góp ý từ quý thầy cô và các bạn để bài báo cáo được hoàn thiện hơn.

Mục lục

Chương 1

Giới thiệu

1.1 Đặt vấn đề

1.1.1 Phân tích bài toán

Trong thời đại công nghệ số phát triển mạnh mẽ, bảo mật thông tin trở thành yếu tố sống còn, đặc biệt trong lĩnh vực ngân hàng – nơi lưu trữ và xử lý hàng triệu giao dịch tài chính mỗi ngày. Việc bảo vệ dữ liệu khách hàng thông qua các phương pháp mã hóa đã trở thành tiêu chuẩn bắt buộc trong hệ thống ngân hàng hiện đại. Tuy nhiên, đa phần sinh viên khi tiếp cận với các thuật toán mã hóa đều cảm thấy khó hiểu, khô khan và thiếu tính trực quan. Điều này gây ra khó khăn trong việc tiếp thu và ứng dụng kiến thức vào thực tiễn.

1.2 Mục tiêu của đề tài

Đề tài xây dựng một trò chơi giáo dục với tên gọi **"Hệ thống mã hóa ngân hàng"**, nhằm mục đích:

- **Mục tiêu 1:** Giúp người học hiểu và thực hành các thuật toán mã hóa cơ bản như Caesar, AES, RSA.
- **Mục tiêu 2:** Mô phỏng các tình huống thực tế trong ngân hàng để tăng tính ứng dụng và thực hành.
- **Mục tiêu 3:** Tạo ra môi trường học tập sinh động, dễ tiếp cận thông qua game hóa kiến thức bảo mật.

1.3 Cấu trúc của báo cáo

Báo cáo gồm các phần như sau:

- Chương 1: Giới thiệu.
- Chương 2: Phân tích yêu cầu và thiết kế ứng dụng
- Chương 3: Kết luận và hướng phát triển

Chương 2

Phân tích yêu cầu và thiết kế ứng dụng

2.1 Mô tả thuật toán

a) Bắt tay (Handshake) đơn giản

Mục đích: Thiết lập kết nối an toàn giữa người dùng và hệ thống ngân hàng.

Quy trình:

1. Client gửi yêu cầu kết nối đến Server (ngân hàng).
2. Server phản hồi bằng khóa công khai (*Public Key*) của nó (thường dùng RSA hoặc ECC).
3. Client kiểm tra chứng chỉ số (nếu có) để xác thực Server.
4. Client tạo khóa phiên (*Session Key*, AES-256) và mã hóa bằng Public Key của Server.
5. Server giải mã bằng Private Key để lấy Session Key.
6. Kết nối an toàn được thiết lập, dữ liệu sau này trao đổi bằng Session Key.

Đảm bảo: Chống nghe lén, tấn công người trung gian (MITM).

b) Xác thực (Ký số và trao đổi khóa)

Mục đích: Xác minh danh tính người dùng và đảm bảo dữ liệu không bị giả mạo.

Thuật toán sử dụng:

- RSA hoặc ECC cho trao đổi khóa và chữ ký số.
- SHA-256 để băm dữ liệu trước khi ký.

Quy trình:

1. Người dùng nhập thông tin (ví dụ: tài khoản, mật khẩu).
2. Hệ thống tạo chữ ký số bằng Private Key của người dùng:

$$\text{Signature} = \texttt{RSA\_Sign(SHA256(data), PrivateKey\_User)}$$

3. Server kiểm tra chữ ký bằng Public Key của người dùng:

$$\text{IsValid} = \texttt{RSA\_Verify(SHA256(data), Signature, PublicKey\_User)}$$

4. Nếu hợp lệ, giao dịch được chấp nhận.

Đảm bảo: Chống giả mạo, xác thực nguồn gốc dữ liệu.

c) Truyền dữ liệu và kiểm tra toàn vẹn

Mục đích: Dữ liệu được mã hóa trong quá trình truyền và đảm bảo không bị sửa đổi.

Thuật toán sử dụng:

- AES-256 ở chế độ CBC hoặc GCM để mã hóa.
- HMAC-SHA256 hoặc GCM để kiểm tra toàn vẹn.

Quy trình:

1. Client mã hóa dữ liệu bằng Session Key (AES):

$$\text{Ciphertext} = \texttt{AES\_Encrypt(Data, SessionKey, IV)}$$

2. Client tạo MAC (Message Authentication Code):

$$\text{MAC} = \texttt{HMAC(SHA256, Data, SecretKey)}$$

3. Gửi `Ciphertext + MAC` đến Server.
4. Server giải mã và kiểm tra MAC:
   - Nếu MAC khớp $\rightarrow$ Dữ liệu toàn vẹn.
   - Nếu không $\rightarrow$ Từ chối giao dịch.

Đảm bảo:

- Dữ liệu không bị lộ: AES.
- Dữ liệu không bị sửa đổi: HMAC.

d) Lưu ý về thuật toán

Tính bảo mật:

- AES-256 đủ mạnh để chống brute-force.
- RSA 2048-bit hoặc ECC 256-bit đảm bảo an toàn cho trao đổi khóa.

Tính toàn vẹn:

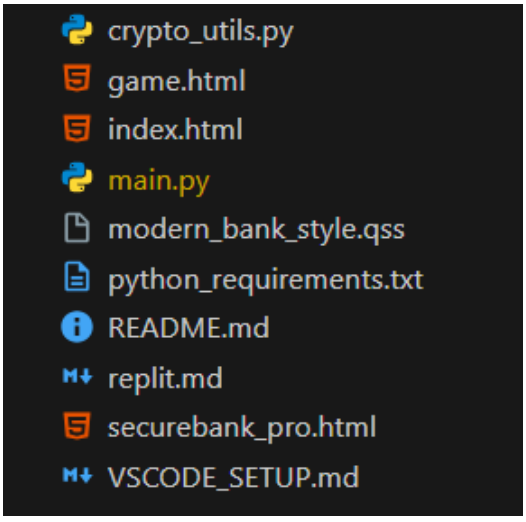
- SHA-256 dùng cho hàm băm.
- HMAC hoặc GCM để xác thực dữ liệu.

Hiệu suất:

- AES nhanh hơn RSA, nên được dùng cho dữ liệu lớn.
- Tránh sử dụng MD5 hoặc SHA-1 do đã có lỗ hổng bảo mật.

2.2 Phân tích mã nguồn

2.2.1 cấu trúc mã nguồn



Các thành phần chính của hệ thống

Các hàm quan trọng

```
def encrypt_aes(data, key):  # Trong crypto_utils.py
def decrypt_aes(ciphertext_b64, key):

# Xử lý giao dịch
```

```
def generate_keys(self):  # Tạo khóa RSA
def encrypt_and_send(self):  # Mã hóa dữ liệu giao dịch
def decrypt_and_verify(self):  # Giải mã dữ liệu
def complete_transaction(self):  # Hoàn thành giao dịch

# Quản lý game
def level_up(self):  # Tăng level
def purchase_upgrade(self, upgrade_key):  # Mua nâng cấp
def show_notification(self, message, success=True):  # Hiển thị thông báo
def game_over(self):  # Xử lý khi game kết thúc
```

Các hàm JavaScript chính:

```
    // Xử lý mã hóa
function generateAESKey(): Tạo khóa AES 256-bit
function encryptData(): Mã hóa dữ liệu giao dịch
function decryptData(): Giải mã dữ liệu

// Xác thực
function generateOTP(): Tạo mã OTP 6 chữ số
function verifyTransaction(): Xác thực giao dịch

// Công cụ bảo mật
function generateHash(): Tạo hash SHA-256
function checkPhishing(): Kiểm tra URL/email lừa đảo

// Quiz bảo mật
function startQuiz(): Bắt đầu quiz
function selectQuizAnswer(index): Chọn đáp án
function submitQuizAnswer(): Nộp đáp án

// Quản lý game
function levelUp(): Tăng level
function updateSecurityLevel(): Cập nhật mức độ bảo mật
function showNotification(message, type): Hiển thị thông báo
function exportData(): Xuất dữ liệu game
function importData(): Nhập dữ liệu game
```

Các hàm mã hóa:

```
    def encrypt_aes(data, key):
    """Mã hóa AES-CBC với padding và IV ngẫu nhiên"""
    key_bytes = key.encode('utf-8')[:16]
    iv = os.urandom(16)
    cipher = AES.new(key_bytes, AES.MODE_CBC, iv)
    encrypted = cipher.encrypt(pad(data.encode('utf-8'), 16)
    return base64.urlsafe_b64encode(iv + encrypted).decode('utf-8')

def decrypt_aes(ciphertext_b64, key):
    """Giải mã AES-CBC với xử lý lỗi"""
    key_bytes = key.encode('utf-8')[:16]
    ciphertext = base64.urlsafe_b64decode(ciphertext_b64)
    iv = ciphertext[:16]
    encrypted_data = ciphertext[16:]
    cipher = AES.new(key_bytes, AES.MODE_CBC, iv)
    return unpad(cipher.decrypt(encrypted_data), 16).decode('utf-8')
```

2.3 Thử nghiệm

2.3.1 Mục tiêu thử nghiệm

Mục tiêu của giai đoạn thử nghiệm hệ thống là để đảm bảo rằng:

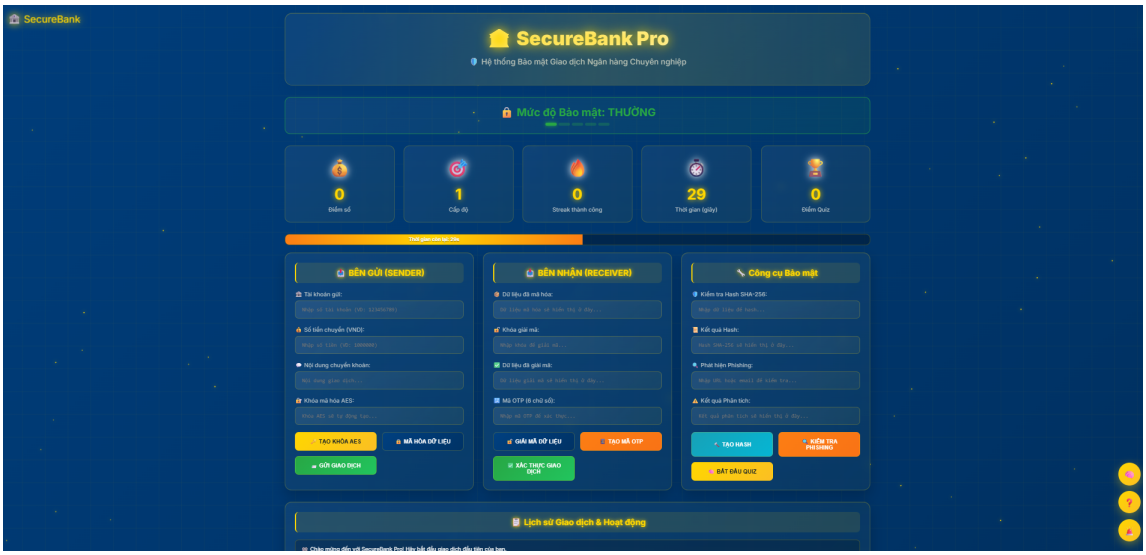
- Toàn bộ hệ thống hoạt động ổn định theo các yêu cầu chức năng và phi chức năng.

- Tất cả các thành phần (frontend, backend, database, API, bảo mật...) tích hợp đúng và hoạt động xuyên suốt.

- Hệ thống có thể xử lý các tình huống thực tế, cả bình thường và bất thường.

- Phát hiện và sửa lỗi trước khi triển khai thực tế cho người dùng cuối.

2.3.2 Nội dung thử nghiệm



Quy trình xử lý giao dịch giữa Sender và Receiver

a. Phía người gửi (Sender)

1. Nhập thông tin tài khoản gửi.
2. Nhập số tiền cần chuyển.
3. Nhập nội dung chuyển khoản.
4. Tạo khoá AES ngẫu nhiên (ví dụ: `QÂ▀P4i7kBRjLTOu▀m7bp2GmXYyguH`).
5. Dữ liệu giao dịch được mã hóa bằng khóa AES.
6. Dữ liệu đã mã hóa được gửi tới phía nhận.

b. Phía người nhận (Receiver)

1. Nhập khóa AES được cung cấp từ người gửi.
2. Nhấn nút **Tạo mã OTP**.
3. Hệ thống sinh ra một mã OTP 6 chữ số.
4. Người dùng nhập mã OTP vào ô xác thực.
5. Nhấn nút **Giải mã dữ liệu** hoặc **Xác thực giao dịch**.
6. Hệ thống kiểm tra mã OTP và giải mã dữ liệu bằng khoá AES.

7. Nếu đúng OTP và đúng khoá: giao dịch được hiển thị hoặc xử lý.

2.4 Đánh giá hiệu quả

2.4.1 Hiệu quả bảo mật

Điểm mạnh về bảo mật

- **Mã hóa mạnh mẽ:**
  - Sử dụng **AES-256** (chế độ CBC) và **RSA-2048** trong `crypto_utils.py`.
  - Mỗi lần mã hóa AES đều sinh IV ngẫu nhiên.
  - Dữ liệu mã hóa được encode theo chuẩn **Base64 URL-safe**.
- **Xác thực đa yếu tố (MFA):**
  - Sử dụng OTP gồm 6 chữ số sinh ngẫu nhiên.
  - Kiểm tra khóa giải mã trước khi hiển thị thông tin giao dịch.
- **Bảo vệ dữ liệu:**
  - Không lưu trữ thông tin nhạy cảm dưới dạng plaintext.
  - Dữ liệu được băm bằng **SHA-256** để kiểm tra toàn vẹn.
- **Phòng chống tấn công:**
  - Tích hợp cơ chế phát hiện phishing cơ bản (kiểm tra URL/email).
  - Giới hạn thời gian thực hiện giao dịch bằng bộ đếm (*timer*).
- **Quản lý khóa hiệu quả:**
  - Khóa AES được tạo mới cho mỗi phiên giao dịch.
  - Khóa RSA có độ dài an toàn (2048-bit).

Hạn chế bảo mật

- **Mã hóa phía client:**
  - Toàn bộ quá trình mã hóa diễn ra ở client (JS/Python), dễ bị chỉnh sửa hoặc phân tích.

- **Lưu trữ khóa không an toàn:**
  - Khóa AES được lưu trong biến JavaScript/Python, dễ bị truy cập bởi mã độc.
- **OTP đơn giản:**
  - OTP là số ngẫu nhiên đơn thuần, không có thời hạn sử dụng rõ ràng.
- **Thiếu xác thực phía server (web version):**
  - Không có lớp API xác thực phía server để xử lý các giao dịch.
- **Có thể bị tấn công dạng *replay*:**
  - Dữ liệu mã hóa có thể bị chặn và gửi lại nếu không có kiểm soát phiên hoặc thời gian.

Khuyến nghị cải thiện

- **Tăng cường xác thực phía server:**
  - Tách phần xử lý giao dịch ra phía backend API.
  - Chỉ cho phép giao dịch sau khi đã được xác thực phía server.
- **Nâng cấp OTP:**
  - Sử dụng **TOTP** (Time-based OTP) thay vì random static OTP.
  - Thiết lập thời gian hết hạn mặc định (ví dụ: 60 giây).
- **Bảo vệ khóa tốt hơn:**
  - Dùng **Web Crypto API** thay vì tự viết mã AES/RSA trong JavaScript.
  - Với phiên bản desktop, nên lưu khóa trong **Secure Enclave** hoặc **Keyring**.
- **Phòng chống tấn công *replay*:**
  - Thêm **nonce** hoặc **timestamp** vào mỗi giao dịch.
  - Áp dụng chữ ký số để xác thực dữ liệu.
- **Kiểm tra đầu vào và bảo mật form:**
  - Kiểm tra tính hợp lệ của input (số tiền, tài khoản, email...).
  - Chống **XSS** bằng cách mã hóa đầu vào.

- **Ghi nhật ký bảo mật:**
  - Ghi log những hành vi đáng ngờ như nhập sai OTP nhiều lần.

2.4.2 Đánh Giá hiệu suất

a) So sánh tốc độ mã hóa/giải mã

Thuật toán	Loại	Tốc độ (trung bình)	Nhận xét
AES-128	Đối xứng	Rất nhanh ($\sim$20 ns/byte)	Phù hợp dữ liệu lớn.
Triple DES	Đối xứng	Chậm hơn ($\sim$3× DES)	Lỗi thời, không phù hợp xử lý thời gian thực.
RSA	Bất đối xứng	Rất chậm (μs/KB)	Chỉ nên dùng cho trao đổi khóa.

b) Kịch bản thực tế

- Với file 1MB:
  - **AES-128**: Mã hóa chỉ mất vài mili giây.
  - **Triple DES**: Mất gấp 3–5 lần thời gian AES.
  - **RSA**: Không phù hợp cho file lớn hơn vài KB do chi phí tính toán.

Chương 3

Kết luận và hướng phát triển

3.1 Phân tích hiệu quả

3.1.1 Phân tích và nhận xét đặc điểm của các thuật toán sử dụng

a. Thuật toán chính: AES (Advanced Encryption Standard)

Sử dụng trong hàm: `encrypt_aes(data, key, mode='CBC')`

Đặc điểm:

- Hỗ trợ nhiều chế độ mã hóa: CBC (mặc định), GCM, CTR,...
- Hỗ trợ các chiều dài khóa:
  - AES-128: 16 byte
  - AES-192: 24 byte
  - AES-256: 32 byte

b. Thư viện liên quan

- `Crypto.Cipher.AES` (từ `PyCryptodome`): Dùng để mã hóa AES.
- `Crypto.Util.Padding`: Cung cấp hàm `pad/unpad` để xử lý dữ liệu vào/ra.
- `Crypto.Random.get_random_bytes`: Tạo IV (Initialization Vector) ngẫu nhiên.
- `hashlib`, `hmac`, `secrets`: Hỗ trợ tạo hàm băm, MAC và sinh khóa ngẫu nhiên an toàn.

c. So sánh hiệu suất và đặc điểm các thuật toán

Thuật toán	Loại mã hóa	Kích thước khóa	Tốc độ	Độ an toàn
AES (CBC)	Đối xứng	128/192/256 bit	Rất nhanh	Rất cao
DES	Đối xứng	56 bit	Trung bình	Thấp
Triple DES	Đối xứng	112/168 bit	Chậm	Trung bình
RSA	Bất đối xứng	1024–4096 bit	Chậm	Rất cao
Blowfish	Đối xứng	32–448 bit	Nhanh	Tốt
ChaCha20	Đối xứng	256 bit	Rất nhanh	Cao
SHA (băm)	Băm	256/512 bit (đầu ra)	Nhanh	Không thể đảo ngược

d. Nhận xét tổng quan

- **AES (CBC)** là lựa chọn phổ biến và hợp lý:
  - Chuẩn hóa bởi NIST.
  - Hỗ trợ nhiều chiều dài khóa.
  - Có thể kết hợp các chế độ như CBC, GCM, CTR để cân bằng giữa hiệu suất và bảo mật.
- **So với RSA:**
  - AES nhanh hơn nhiều do là mã hóa đối xứng.
  - RSA phù hợp cho trao đổi khóa và chữ ký số, không dùng mã hóa dữ liệu lớn.
- **So với ChaCha20:**
  - AES hiệu quả trên phần cứng (nhờ hỗ trợ AES-NI).
  - ChaCha20 phù hợp hơn với thiết bị di động hoặc môi trường không hỗ trợ phần cứng AES.

3.1.2 Đề xuất cải tiến

a. Cải tiến về bảo mật

- **Chuyển sang AES-GCM thay vì AES-CBC:**
  - **AES-GCM** hỗ trợ mã hóa có xác thực (*authenticated encryption*).
  - Cho phép phát hiện các chỉnh sửa dữ liệu bất hợp pháp.

  - Không cần HMAC riêng biệt vì GCM tích hợp xác thực.

- **Bảo vệ khóa và IV:**
  - Không `hardcode` khóa trong mã nguồn.
  - Lưu trữ khóa trong môi trường bảo mật như biến môi trường hoặc dịch vụ **key vault** (VD: AWS KMS, Azure Key Vault).
- **Sử dụng HMAC nếu tiếp tục dùng AES-CBC:**
  - Dùng thuật toán **HMAC-SHA256** để xác thực dữ liệu sau khi mã hóa.
  - Giúp phát hiện dữ liệu bị sửa đổi trong quá trình truyền.
- **Sử dụng Salt và KDF (Key Derivation Function):**
  - Áp dụng các thuật toán như **PBKDF2**, **bcrypt** hoặc **scrypt** để sinh khóa từ mật khẩu người dùng.
  - Tăng độ an toàn, ngăn tấn công brute-force và rainbow table.

Tài liệu tham khảo

[1] Stallings, W. (2021). *Cryptography and Network Security: Principles and Practice* (8th Edition). Pearson.

[2] Paar, C., & Pelzl, J. (2010). *Understanding Cryptography: A Textbook for Students and Practitioners*. Springer.

[3] National Institute of Standards and Technology (NIST). (2001). *FIPS PUB 197: Advanced Encryption Standard (AES)*.
https://csrc.nist.gov/publications/detail/fips/197/final

[4] Nakamoto, S. (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System*.
https://bitcoin.org/bitcoin.pdf

[5] OWASP Foundation. (2023). *OWASP Cryptographic Storage Cheat Sheet*.
https://cheatsheetseries.owasp.org/cheatsheets/Cryptographic_Storage_Cheat_Sheet.html