Chapter 1

Introduction

1.1 Problem Statement

The project addresses the challenge of securely storing and handling **credit card numbers** in applications that require the **preservation of data format**. Traditional encryption methods, like AES, alter the structure of data, making them unsuitable for fields like credit card numbers which must retain their **length and numeric constraints**.

This application uses **Format-Preserving Encryption (FPE)** — specifically the **FE1 algorithm** (as implemented in the `Worldpay java-fpe` library) — to:

- Encrypt 16-digit card numbers into 16-digit cipher text.
- Decrypt encrypted values back to their original form.
- Perform validation (via the Luhn algorithm) and mask sensitive data.
- Log all activity for auditing purposes.

> **Key Security Concern:** Maintain format while ensuring resistance to cryptographic attacks such as **ciphertext-only attacks**, **known-plaintext attacks**, and potential **brute-force vulnerabilities**.

1.2Motivation for Choosing the Problem Statement

The project is motivated by the following practical needs:

- **Legacy System Compatibility:** Many existing systems require data in fixed-length numeric formats (e.g., 16-digit fields). Traditional encryption breaks this requirement.
- **Regulatory Compliance:** Regulations such as **PCI-DSS** mandate strong encryption for cardholder data. FPE offers compliance while maintaining usability.

- **Enhanced User Security:** By integrating **Luhn validation**, **masked output**, and **audit logging**, the application aims to reduce misuse and data leakage.

- **Educational Value:** Implementing FPE in a Java GUI provides a valuable opportunity to combine cryptography, user interface development, and secure software design.

1.3 Aspects of the Algorithm Chosen (FE1 - Format-Preserving Encryption)

Key Features:

- **Preserves Format:** Encrypts numeric strings without changing their length or character set (ideal for credit card numbers).

- **SecureRandom Key Generation:** Each session generates a secure encryption key, ensuring unpredictability.

- **Tweak Usage:** A fixed tweak value is used to add diversity to the encryption context.

- **Radix Handling:** Works specifically with base-10 (decimal) numbers to match credit card formats.

Chapter 2

Methodology

Implementation Procedure: Step-by-Step

1. Set Up the Project Environment

- Create a Java Swing GUI project.
- Import the **Worldpay `java-fpe` library**.
- Configure the classpath to include `FE1` class.
- Set up initial variables: encryption key, tweak, and modulus.

2. Build the GUI Components

- Input field for 16-digit credit card number or encrypted text.
- Result display field (masked or full).
- Buttons:
    - **Encrypt**
    - **Decrypt**
    - **Clear**
- Checkboxes:
    - **Luhn Validation**
    - **Masked Output**
    - **Real-time Encryption**
- Tooltips and input restrictions for user guidance.

3. Encrypt / Decrypt Logic

- **Encrypt button clicked:**
    - If Luhn validation is enabled, verify the card number.
    - Encrypt using FE1 algorithm.
    - Format the encrypted result (16 digits).
- **Decrypt button clicked:**
    - Decrypt using FE1 algorithm.
    - Show masked or full number based on checkbox.

4. Real-Time Encryption (Optional)

- Listen to key events in the input field.
- Automatically encrypt each character as it's typed (optional feature toggle).

5. Audit Logging

- For each encryption/decryption:
    - Log timestamp, operation type, and masked card number.
    - Save to `audit_log.txt` file.
- Ensure the file is created/appended securely.

6. Session Key Management

- Generate session-based encryption key using `SecureRandom`.
- Store temporarily for use in the current session.
- (Optional future scope): Save/load key for consistent behavior across sessions.

7. Testing and Validation

- Test valid and invalid card numbers.
- Check Luhn validation functionality.
- Validate encrypted output format (should always be 16 digits).
- Confirm audit logs are correct.

Chapter 4
4.1 Conclusion

This project successfully demonstrates the use of **Format-Preserving Encryption (FPE)** to secure credit card numbers in a desktop GUI environment built with Java Swing. By using the **FE1 scheme** from the Worldpay Java-FPE library, the application ensures that encrypted card numbers retain the same length and numeric format—an essential feature for legacy systems and compliance with financial standards.

Key takeaways from the implementation:

- The **Luhn algorithm** integration ensures only valid card numbers are encrypted.
- **Masked output** protects sensitive decrypted data from onlookers.
- **Audit logging** provides accountability and traceability for all cryptographic actions.
- **Real-time encryption** improves interactivity and responsiveness for users.

The application balances **security, usability, and regulatory requirements**, making it a practical tool for secure credit card data handling in small-scale or educational environments.

4.2 Future Work

To enhance the robustness and real-world applicability of the project, the following improvements are suggested:

1. **Persistent Key Management:**
    - Add the ability to **save/load encryption keys** securely, ensuring encrypted data remains decryptable across sessions.
2. **Batch File Encryption:**
    - Extend support to process **CSV files** containing multiple card numbers.
3. **Authentication System:**

- Implement a **login mechanism** to control access to encryption/decryption operations.

4. **Audit Log Viewer:**
    - Create a **GUI-based viewer** to browse and filter log entries.
5. **User-Defined Tweak and Modulus:**
    - Allow customization of tweak/modulus values to strengthen cryptographic diversity.
6. **Advanced Attack Testing:**
    - Integrate deeper analysis tools or attack simulations as part of an extended **AAT module**.