# **CHAPTER THREE**

## **METHODOLOGY**

### **3.1 OVERVIEW**

This chapter outlines the methodology adopted in designing and implementing a secure, efficient, and user-friendly file encryption system. The development process was guided by the project’s objectives: to integrate strong encryption algorithms with an intuitive user interface and effective key management. The methodology encompasses requirement gathering, system design, tool selection, and evaluation strategies to ensure that the resulting system meets security, usability, and performance standards.

### **3.2 REQUIREMENT GATHERING**

The functional and non-functional requirements were established through a review of existing file encryption tools, literature on cryptographic standards, and consultations with cybersecurity experts. The primary focus was to identify key features that would provide robust security while ensuring usability for non-technical users.

**Functional Requirements:**

* Ability to encrypt and decrypt files using strong cryptographic algorithms.
* Support for common file types such as .txt, .pdf, .docx, .jpg, and .png.
* Secure key generation, storage, and retrieval.
* Simple user interface for selecting files and performing encryption tasks.

**Non-Functional Requirements:**

* Minimal processing time for encryption and decryption.
* Compatibility with Windows and Linux platforms.
* Lightweight and intuitive application design.
* Resistance to brute-force and replay attacks.

### **3.3 SYSTEM DESIGN AND SPECIFICATION**

#### **3.3.1 Architecture Overview**

The system adopts a modular architecture, comprising four main components:

1. **Cryptographic Core**: Implements AES (Advanced Encryption Standard) for data encryption and RSA for key wrapping and exchange.
2. **User Interface (UI)**: Provides a graphical environment for users to interact with the system.
3. **Key Management Module**: Handles secure key generation, storage (encrypted locally), and retrieval.
4. **File I/O Handler**: Manages file selection, reading, and writing of encrypted or decrypted outputs.

#### **3.3.2 Data Flow**

1. User selects a file via the interface.
2. The system generates or loads an AES key for file encryption.
3. The AES key is encrypted using an RSA public key and stored securely.
4. The file is encrypted with the AES key and saved as a .enc file.
5. For decryption, the RSA private key is used to retrieve the AES key and decrypt the file.

#### **3.3.3 Security Design**

* AES-256 in CBC mode is used for encrypting file contents.
* RSA-2048 is used for encrypting AES keys.
* A secure random number generator is used for key generation.
* Encrypted keys are stored in local encrypted key vaults using password-based access control.

### **3.4 SYSTEM DEVELOPMENT TOOLS**

| **Category** | **Tool** | **Description** |
| --- | --- | --- |
| Programming Language | Python | Chosen for its readability and strong cryptographic libraries (e.g., cryptography, PyCryptodome) |
| GUI Framework | Tkinter | For building a simple, cross-platform graphical interface |
| Cryptography Libraries | PyCryptodome, Cryptography.io | Provide AES and RSA implementations |
| Development Environment | Visual Studio Code | IDE for writing and debugging code |
| Version Control | Git | Used for version tracking and collaboration |
| Operating System | Windows/Linux | Ensures compatibility across multiple platforms |

### **3.5 IMPLEMENTATION PROCEDURE**

#### **3.5.1 Encryption Workflow**

* User selects a file via the GUI.
* The system generates a random AES key and IV (Initialization Vector).
* The file is read in chunks and encrypted using AES-CBC.
* The AES key is encrypted using RSA and saved in a separate .key file.
* The encrypted file is saved with a. enc extension.

#### **3.5.2 Decryption Workflow**

* The user selects the .enc file and the corresponding .key file.
* The RSA private key is used to decrypt the AES key.
* The encrypted file is read and decrypted in chunks.
* The original file is restored and saved to the user's desired location.

### **3.6 TESTING AND EVALUATION STRATEGY**

#### **3.6.1 Test Case Scenarios**

| **Test Case ID** | **Test Description** | **Expected Result** |
| --- | --- | --- |
| TC-1 | Encrypt .txt file | Encrypted .enc file created |
| TC-2 | Decrypt .enc file | Original .txt file restored |
| TC-3 | Invalid decryption key | Decryption fails gracefully |
| TC-4 | Encrypt large file (100MB+) | File encrypted successfully within acceptable time |
| TC-5 | User cancels operation mid-process | Process aborts without corruption |

#### **3.6.2 Performance Metrics**

To evaluate the system, the following metrics were used:

* **Encryption/Decryption Time**: Time taken to process files of different sizes.
* **Key Management Security**: Resistance to unauthorized access to stored keys.
* **User Feedback**: Usability and interface clarity rated by test users.
* **Error Handling**: Graceful responses to file format errors, key mismatches, or user interruptions.

#### **3.6.3 Benchmarking**

The system was benchmarked against AxCrypt and Cryptomator for:

* Speed of encryption/decryption.
* Size of encrypted files.
* User experience based on standardized usability questionnaires.