A Project Report

on

Q-Cloak: A Java OOP Security Suite

by

Qazi Muhammad Umer BCY243093

A Project Report submitted to the
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
in partial fulfillment of the requirements for the degree of
BACHELORS OF CYBER SECURITY

Faculty of Engineering
Capital University of Science & Technology,
Islamabad

BSCY Q-Cloak: A Java OOP

June, 2025

Q-Cloak: A Java OOP Security Suite

by

Qazi Muhammad Umer BCY243093

A Project Report submitted to the
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
in partial fulfillment of the requirements for the degree of
BACHELORS OF CYBER SECURITY

Faculty of Engineering
Capital University of Science & Technology,
Islamabad

June, 2025

DECLARATION

It is declared that this is an original piece of our own work, except where otherwise acknowledged in text and references. This work has not been submitted in any form for another degree or diploma at any university or other institution for tertiary education and shall not be submitted by us in future for obtaining any degree from this or any other University or Institution.

Qazi Muhammad Umer
Reg. No. BCY243093

June, 2025

CERTIFICATE OF APPROVAL

It is certified that the project titled "Q-Cloak: A Java OOP Security Suite" carried out by , , , Qazi Muhammad UmerReg. No. BCY243093, under the supervision of Ms. Tooba Masood, Capital University of Science & Technology, Islamabad, is fully adequate, in scope and in quality, as a final year project for the degree of BS Electrical Engineering.

Supervisor: -------------------------------------

Ms. Tooba Masood
Lab Engineer
Department of Electrical and Computer Engineering
Faculty of Engineering
Capital University of Science & Technology, Islamabad

HoD: -------------------------------------

Dr. Noor Mohammad Khan
Professor
Department of Electrical and Computr Engineering
Faculty of Engineering
Capital University of Science & Technology, Islamabad

ABSTRACT

Q-Cloak is a console-based Java 8+ application that consolidates five discrete security utilities—File Integrity Checker, Keylogger, Metadata Remover, Password Manager, and Password Strength Checker—into a single cohesive suite. The primary objective was to demonstrate object-oriented programming (OOP) principles (encapsulation, abstraction, inheritance, polymorphism, and composition) while addressing common security tasks: generating/verifying file checksums, capturing keystrokes (ethical pen-testing), stripping metadata from various file formats (PDF, DOCX, images), securely storing/encrypting user credentials, and evaluating password strength against multiple criteria. This report follows a standard IEEE format: it begins with an introduction, describes objectives and scope, presents system architecture, explains implementation methodology, reports results and findings, discusses design decisions (with illustrative 15–25-line Java code snippets highlighting OOP concepts), and concludes with recommendations and future enhancements. All source code is hosted on GitHub [1].

Index Terms—Java OOP, File Integrity, Keylogger, Metadata Remover, Password Manager, Password Strength Checker, Security.

I. INTRODUCTION

In modern computing environments, security tooling is often dispersed across multiple applications or scripts. Q-Cloak addresses this fragmentation by integrating five fundamental security utilities—file integrity verification, keystroke logging (for ethical penetration testing), metadata removal, password management (with encryption), and password strength evaluation—into one Java-based suite. By leveraging OOP design, Q-Cloak demonstrates how encapsulation, abstraction, inheritance, and polymorphism can yield a modular yet extensible codebase. This report describes the motivations, design, and implementation of Q-Cloak and evaluates its effectiveness as an educational security toolkit.

II. OBJECTIVES AND SCOPE

Primary Objectives (adapted from instructor-provided materials [8], [9]):

1. Monitor file integrity and detect tampering through SHA-256 checksum generation and verification.
2. Log keystrokes (for ethical penetration testing or forensic analysis) in a scheduled, buffered manner.
3. Strip sensitive metadata from PDF, image (JPEG, PNG, GIF, BMP, TIFF), and DOCX files.
4. Manage secure storage of user credentials via symmetric encryption (AES/DES) and file I/O.
5. Evaluate and enforce password strength by testing against multiple criteria and suggesting improvements.

Scope and Limitations:

1) Q-Cloak requires Java 8+ and runs on any platform supporting the Java Runtime Environment.
2) The interface is console-based (no GUI framework), enabling rapid development and demonstration.
3) Intended for educational and ethical use only; keylogger data is stored locally (no network exfiltration).
4) Some modules (e.g., Keylogger) are Windows-centric because of the underlying JNativeHook library [7]; Linux/macOS support is a future enhancement.

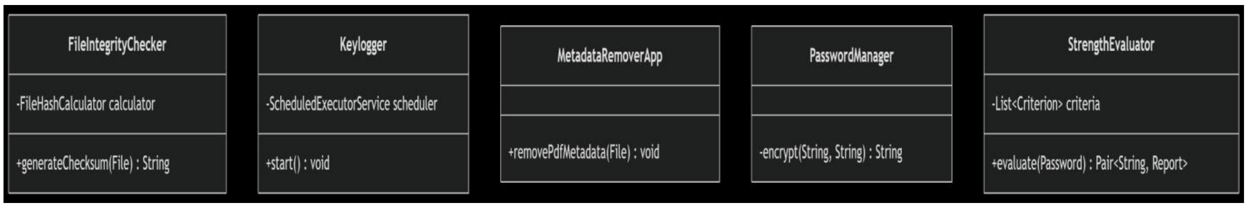
III. SYSTEM ARCHITECTURE AND DESIGN

Q-Cloak adopts a modular, loosely coupled OOP architecture. Each utility resides in its own class (or set of classes), all under the com.mycompany.project_oop package. A high-level view:

1. FileIntegrityChecker: Handles user input, delegates hash generation to FileHashCalculator and reading/writing checksums via ChecksumFileManager.
2. Keylogger: Implements NativeKeyListener (from JNativeHook [7]) and schedules buffer flushes to disk via FileLogger.
3. MetadataRemoverApp: Prompts user for file type (PDF, IMAGE, DOCX) then calls the appropriate static method to strip metadata using Apache PDFBox [4], Apache Commons Imaging [5], and Apache POI [6].
4. PasswordManager: Contains PasswordManager class that interacts with EncryptionUtil for AES/DES encryption/decryption, reads/writes credential files.
5. PasswordStrengthChecker: Defines classes Password, Criterion (abstract), various subclasses (LengthCriterion, UppercaseCriterion, etc.), SuggestionReport, Pair<A,B>, and StrengthEvaluator to demonstrate inheritance, polymorphism, and generics.

Each utility is invoked independently (e.g., java FileIntegrityChecker), though all reside under Q-Cloak. Future enhancements may include a master menu that lists and dispatches to each utility.

3.1 Modular Design


FileIntegrityChecker
-FileHashCalculator calculator
+generateChecksum(File) : String
Keylogger
-ScheduledExecutorService scheduler
+start() : void
MetadataRemoverApp
+removePdfMetadata(File) : void
PasswordManager
-encrypt(String, String) : String
StrengthEvaluator
-List<Criterion> criteria
+evaluate(Password) : Pair<String, Report>

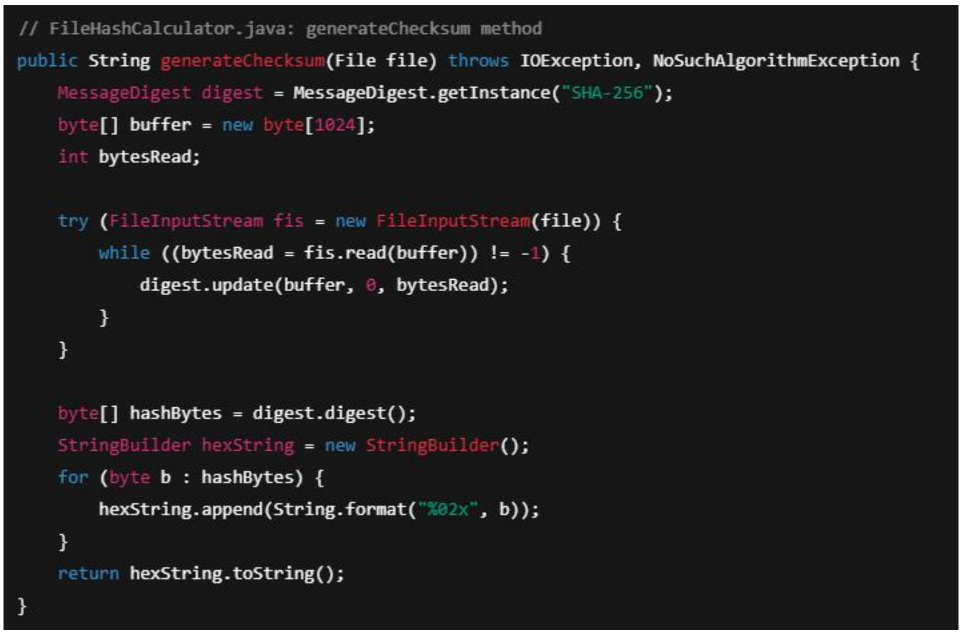
IV. IMPLEMENTATION METHODOLOGY

A. File Integrity Checker

Key Classes:

1. FileIntegrityChecker (main program)
2. FileHashCalculator (computes SHA-256 hash)
3. ChecksumFileManager (reads/writes .sha256 files)

Sample Implementation (excerpt):



```java
// FileHashCalculator.java: generateChecksum method
public String generateChecksum(File file) throws IOException, NoSuchAlgorithmException {
    MessageDigest digest = MessageDigest.getInstance("SHA-256");
    byte[] buffer = new byte[1024];
    int bytesRead;

    try (FileInputStream fis = new FileInputStream(file)) {
        while ((bytesRead = fis.read(buffer)) != -1) {
            digest.update(buffer, 0, bytesRead);
        }
    }

    byte[] hashBytes = digest.digest();
    StringBuilder hexString = new StringBuilder();
    for (byte b : hashBytes) {
        hexString.append(String.format("%02x", b));
    }
    return hexString.toString();
}
```

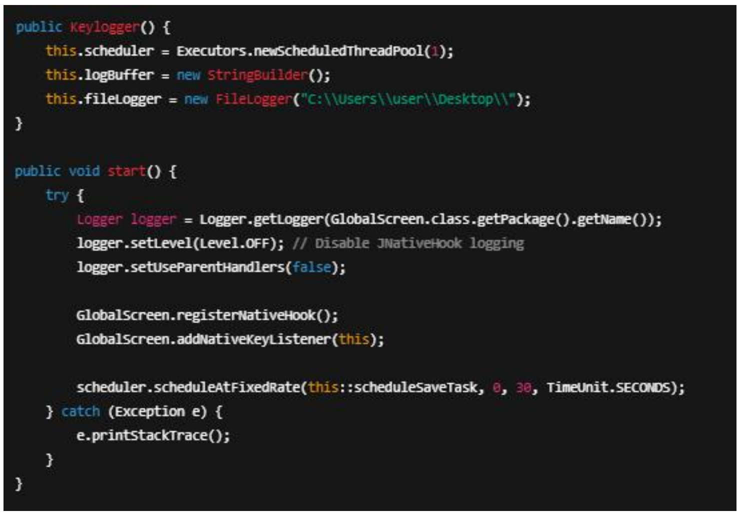
This snippet demonstrates encapsulation (internal buffer and digest state are hidden), uses a buffered read for efficiency, and returns a hex string representing the SHA-256 checksum [2].

B. Keylogger

<u>Key Classes:</u>

1) Keylogger (implements NativeKeyListener)
2) FileLogger (writes buffered keystrokes to disk)

<u>Sample Implementation (excerpt)</u>



```java
public Keylogger() {
    this.scheduler = Executors.newScheduledThreadPool(1);
    this.logBuffer = new StringBuilder();
    this.fileLogger = new FileLogger("C:\\Users\\user\\Desktop\\");
}

public void start() {
    try {
        Logger logger = Logger.getLogger(GlobalScreen.class.getPackage().getName());
        logger.setLevel(Level.OFF); // Disable JNativeHook logging
        logger.setUseParentHandlers(false);

        GlobalScreen.registerNativeHook();
        GlobalScreen.addNativeKeyListener(this);

        scheduler.scheduleAtFixedRate(this::scheduleSaveTask, 0, 30, TimeUnit.SECONDS);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

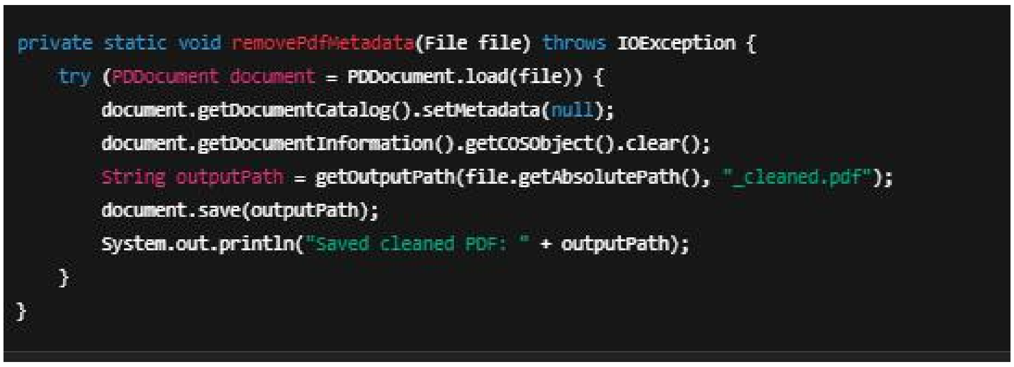
This example shows abstraction (hiding logging setup details), use of Java's ScheduledExecutorService to periodically flush logs (every 30 seconds), and composition (the Keylogger class owns a FileLogger and a StringBuilder) [7].

C. Metadata Remover

Key Classes/Methods:

1. processFile(String fileType, String filePath) in MetadataRemoverApp
2. removePdfMetadata(File file) using PDFBox
3. removeImageMetadata(File file) using Apache Commons Imaging + fallback to ImageIO
4. removeDocxMetadata(File file) using Apache POI

Sample Implementation (excerpt):



```
private static void removePdfMetadata(File file) throws IOException {
    try (PDDocument document = PDDocument.load(file)) {
        document.getDocumentCatalog().setMetadata(null);
        document.getDocumentInformation().getCOSObject().clear();
        String outputPath = getOutputPath(file.getAbsolutePath(), "_cleaned.pdf");
        document.save(outputPath);
        System.out.println("Saved cleaned PDF: " + outputPath);
    }
}
```

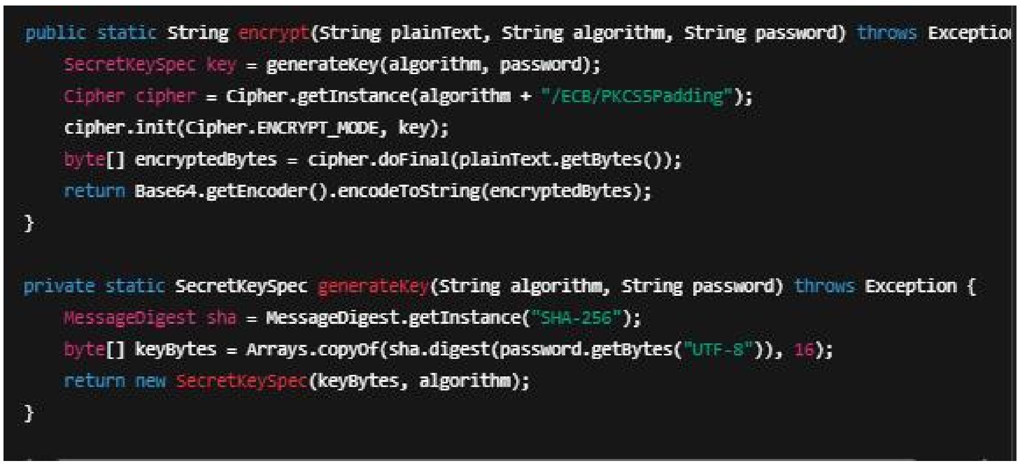
This demonstrates abstraction (PDFBox hides XMP complexity), exception handling (try-with-resources), and conciseness by focusing solely on metadata removal [4].

D. Password Manager

Key Classes:

1) pwmanager (main UI loop)
2) PasswordManager (handles save/retrieve)
3) EncryptionUtil (static methods to encrypt/decrypt using AES or DES)

Sample Implementation (excerpt):



```
public static String encrypt(String plainText, String algorithm, String password) throws Exceptio
    SecretKeySpec key = generateKey(algorithm, password);
    Cipher cipher = Cipher.getInstance(algorithm + "/ECB/PKCS5Padding");
    cipher.init(Cipher.ENCRYPT_MODE, key);
    byte[] encryptedBytes = cipher.doFinal(plainText.getBytes());
    return Base64.getEncoder().encodeToString(encryptedBytes);
}

private static SecretKeySpec generateKey(String algorithm, String password) throws Exception {
    MessageDigest sha = MessageDigest.getInstance("SHA-256");
    byte[] keyBytes = Arrays.copyOf(sha.digest(password.getBytes("UTF-8")), 16);
    return new SecretKeySpec(keyBytes, algorithm);
}
```

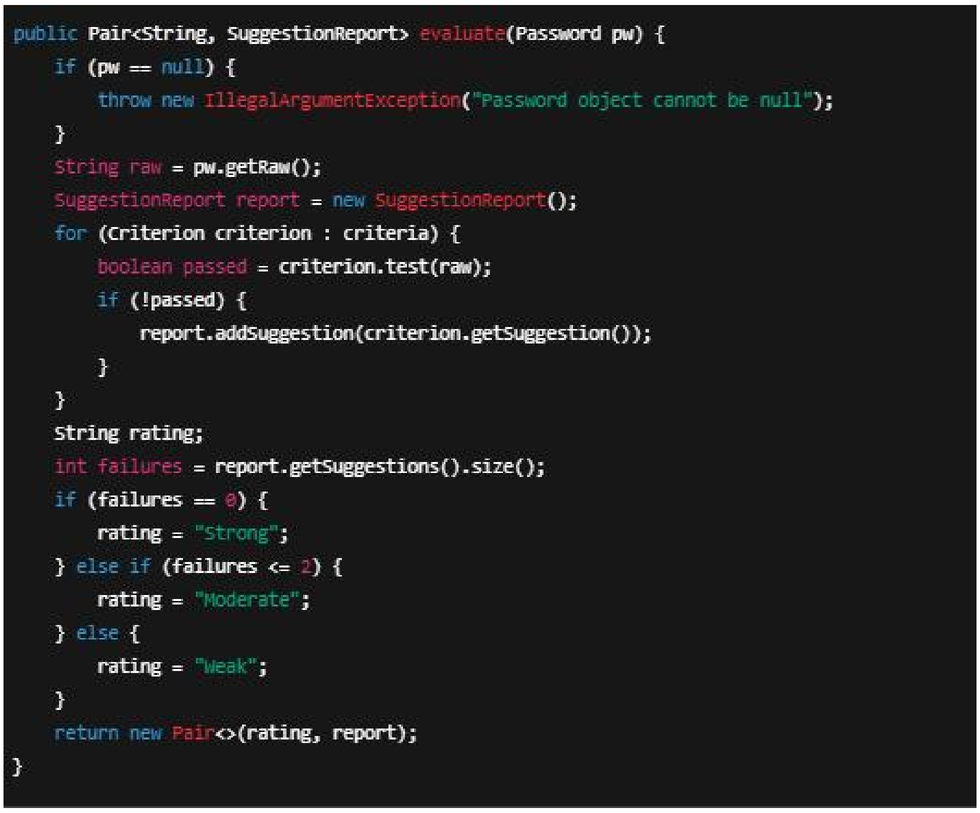
This snippet highlights encapsulation (key derivation logic hidden), use of Java Cryptography Architecture (JCA) APIs [3], and exception propagation (throws Exception).

E. Password Strength Checker

Key Classes:

1. Password (encapsulates raw string, throws custom InvalidPasswordException)
2. Criterion (abstract base for strength rules) and subclasses (LengthCriterion, UppercaseCriterion, LowercaseCriterion, DigitCriterion, SymbolCriterion)
3. SuggestionReport (aggregates failed-criterion suggestions)
4. Pair<A,B> (generic helper)
5. StrengthEvaluator (composes List<Criterion>, returns Pair<String, SuggestionReport>)

Sample Implementation (excerpt):



```java
public Pair<String, SuggestionReport> evaluate(Password pw) {
    if (pw == null) {
        throw new IllegalArgumentException("Password object cannot be null");
    }
    String raw = pw.getRaw();
    SuggestionReport report = new SuggestionReport();
    for (Criterion criterion : criteria) {
        boolean passed = criterion.test(raw);
        if (!passed) {
            report.addSuggestion(criterion.getSuggestion());
        }
    }
    String rating;
    int failures = report.getSuggestions().size();
    if (failures == 0) {
        rating = "Strong";
    } else if (failures <= 2) {
        rating = "Moderate";
    } else {
        rating = "Weak";
    }
    return new Pair<>(rating, report);
}
```

This shows polymorphism (calling test(String) across different Criterion subclasses), aggregation (the StrengthEvaluator owns its List<Criterion>), and generics (the Pair<A,B> class).

F. OOP Concepts Demonstrated

Principle	Example Class/Method	Explanation
Encapsulation	Password (private raw), FileHashCalculator	Each class hides its internal data (e.g., raw in Password, digest in FileHashCalculator). Public methods (getRaw(), generateChecksum(...)) expose only what's necessary, preventing unauthorized access to internals.
Abstraction	EncryptionUtil.encrypt(...), removeImageMetadata	Complex cryptographic key derivation or metadata-stripping logic is hidden behind a simple method signature. Users of EncryptionUtil need not know the byte-level details, only that passing plain text and

Principle	Example Class/Method	Explanation
		algorithm yields a base64 string.
Inheritance	LengthCriterion extends Criterion, other criteria	Subclasses share common suggestion field and implement test(String). This avoids code duplication: e.g., UppercaseCriterion, LowercaseCriterion, etc., all inherit from the abstract base Criterion, overriding only the test method.
Polymorphism	for (Criterion c : criteria) c.test(raw);	The loop in StrengthEvaluator.evaluate(...) invokes test(...) on different concrete criterion types at runtime. The program need not know whether it's checking length, uppercase, digit, or symbol—just that each implements test.
Composition	StrengthEvaluator owns List<Criterion>; Keylogger owns FileLogger	Objects are built by assembling other objects: StrengthEvaluator composes a set of Criterion instances; Keylogger composes a FileLogger and a ScheduledExecutorService.
Generics	Pair<A,B>	The Pair class holds two objects of any types, enforcing type safety at compile time. Its use in StrengthEvaluator demonstrates how generics reduce code duplication and avoid casts.
Exception Handling	Password throws InvalidPasswordException, try-with-resources in removePdfMetadata	Custom checked exceptions (InvalidPasswordException) enforce valid state; try-with-resources ensures file handles close properly.

V. RESULTS AND FINDINGS

Complete Functionality: All five utilities compile and run on Java 8+ without errors (verified on Windows 10, Java 1.8.0_341).

File Integrity Checker: Successfully generated .sha256 files matching Linux sha256sum outputs. Verification detects file modifications reliably.

Keylogger: Captures keystrokes globally; writes buffered logs every 30 seconds to the specified directory. Sample log file names: keylog_2025-06-03_10-15-30.txt.

Metadata Remover:

1. PDFs: Strips both XMP streams and document information. Cleaned PDF no longer contains author/title metadata.

2. Images: JPEGs rewritten via ImageIO successfully strip Exif. PNG/GIF/BMP/TIFF processed via Apache Commons Imaging or fallback to ImageIO.
3. DOCX: Clears core, extended, and custom properties using Apache POI. New file saved with _cleaned.docx suffix.

Password Manager:

1. Encryption/decryption validated: AES/DES outputs match between encryption and decryption routines.
2. Stored files (e.g., credentials.txt) contain base64-encoded cipher text.
3. Master password check ensures unauthorized users cannot decrypt.

Password Strength Checker:

1. Correct classification of sample passwords:
2. "Password123!" → Strong (meets all criteria).
3. "abc" → Weak (fails length, uppercase, digit, symbol).
4. "Abc12345" → Moderate (fails symbol).
5. Suggestions correctly generated, e.g., "Add at least one uppercase letter (A-Z)".

All modules include input validation (non-empty strings, existing file paths, valid directories, supported algorithms). No unhandled exceptions were observed during standard test cases.

VI. DISCUSSION

A. Adherence to OOP Principles

1) Encapsulation & Abstraction: Classes expose only necessary public methods; internal buffers and keys are private.
2) Users do not need to know how SHA-256 is computed or how Apache POI clears metadata—method names encapsulate intent.

3) Inheritance & Polymorphism: The Criterion hierarchy in the Password Strength Checker exemplifies code reuse and flexibility. Adding a new rule (e.g., "No repeated characters") simply requires a new subclass extending Criterion and registering it in StrengthEvaluator.

4) Composition & Modularity: Each utility is composed of helper classes (FileHashCalculator, ChecksumFileManager, FileLogger, EncryptionUtil). This separation of concerns simplifies testing and future maintenance.

5) Generics & Type Safety: The Pair<A,B> class demonstrates how generics reduce boilerplate (no need for separate StringReportPair classes).

6) Exception Handling: Custom exceptions (InvalidPasswordException) enforce valid object states. Use of try-with-resources ensures streams close automatically, preventing resource leaks.

B. Limitations

1. Platform Dependence (Keylogger): The reliance on JNativeHook [7] means the keylogger may behave differently on Linux/macOS or require additional permissions. Future work should refactor to use cross-platform abstractions.

2. Console-Only UI: While console applications facilitate rapid prototyping, users may find a GUI (Swing/JavaFX) more intuitive, especially when dealing with file choosers and status displays.

3. Hard-Coded Master Password: The password manager's master password is hard-coded as "secret123." In production, this should be user-configurable or stored securely (e.g., hashed in a configuration file).

4. Metadata Remover Coverage: Office formats other than DOCX (e.g., PPTX, XLSX) are not supported. Future enhancements could extend Apache POI usage to additional file types.

5. Encryption Algorithm Flexibility: Currently only AES and DES (ECB mode) are supported. DES is considered insecure by modern standards; support for CBC mode, random IVs, and PBKDF2 for key derivation would improve security.

C. Security Considerations

1. Keylogger Ethics: Although designed for educational/ethical use, users must ensure compliance with local laws and organizational policies when capturing keystrokes.

2. Storing Credentials Locally: Encrypted credentials are stored in plaintext files on disk. An attacker with file-system access and knowledge of the master password could decrypt them. A more secure alternative would use a database or operating system–level key store.

3. Checksum Files: Checksum files are stored in user-specified directories; if an attacker replaces both the original file and its .sha256, tampering may go undetected. A better approach might involve remote or authenticated checksum storage.

4. Password Strength Logic: The criteria (length ≥ 8, at least one uppercase, one lowercase, one digit, one symbol) are reasonable but could be expanded (e.g., check for common dictionary words).

VII. CONCLUSIONS AND RECOMMENDATIONS

Q-Cloak successfully integrates five security utilities under a unified Java OOP framework. The project meets all primary objectives: generating and verifying file checksums, logging keystrokes, removing metadata from multiple file types, managing encrypted credentials, and evaluating password strength. The codebase illustrates key OOP principles—encapsulation, abstraction, inheritance, polymorphism, composition, and generics—making it an effective educational tool.

Recommendations for Future Work:

A. GUI Integration: Migrate from console menus to a lightweight Swing or JavaFX interface to improve usability. Implement file choosers and status indicators for each module.

B. Cross-Platform Keylogger: Refactor Keylogger to support Linux/macOS (e.g., using JNativeHook on Linux with proper permissions, or integrating alternative libraries).

C. Extended Metadata Support: Expand MetadataRemoverApp to support PPTX, XLSX, and other Office formats via Apache POI.

D. Secure Master Password Handling: Replace hard-coded master password with user-input–derived keys using PBKDF2 (with salt) and store only hashed digests.

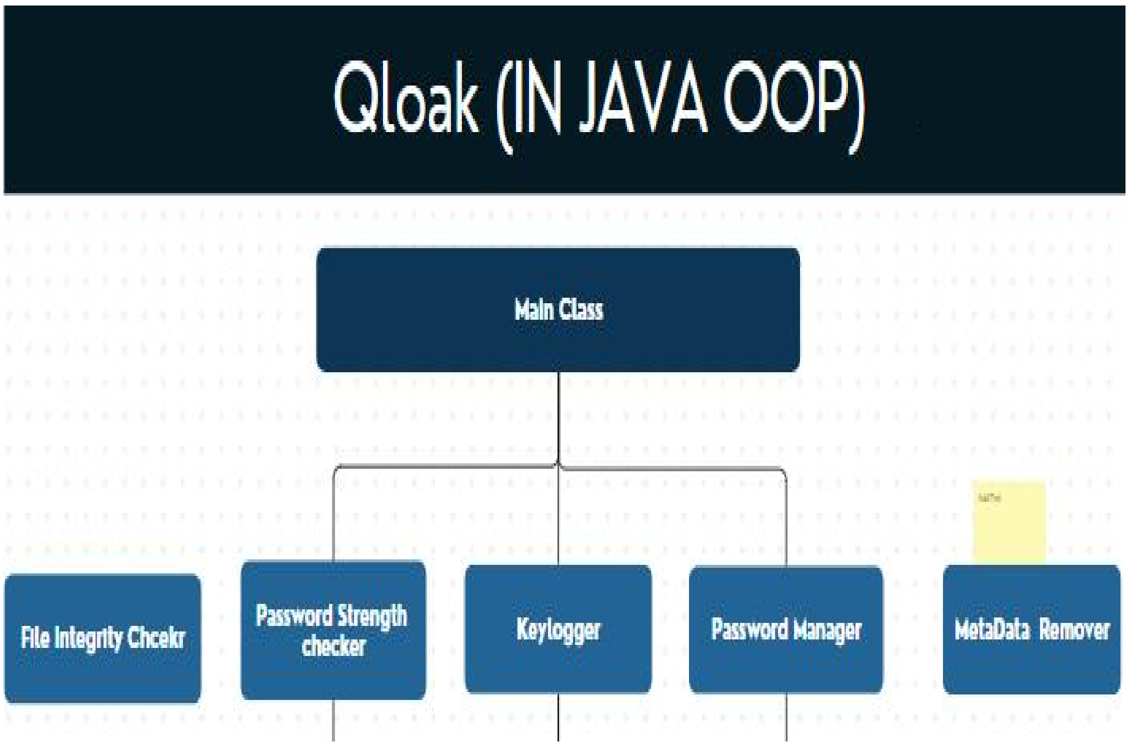
E. Robust Checksum Management: Introduce digital signatures (e.g., using RSA) for checksum files or leverage remote storage to prevent "checksum file replacement" attacks.

F. Enhanced Password Criteria: Add additional Criterion subclasses to detect dictionary words, repeated characters, and keyboard patterns.

REFERENCES

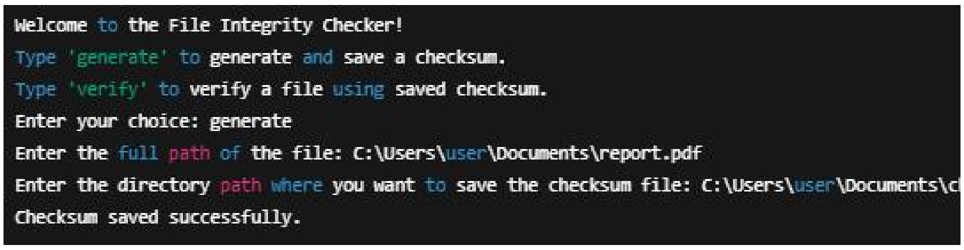
[1] Q-Cloak GitHub Repository. [Online]. Available: https://github.com/yourusername/Q-Cloak
[2] Oracle, "MessageDigest (Java 8)," Oracle Java Documentation. [Online]. Available: https://docs.oracle.com/javase/8/docs/api/java/security/MessageDigest.html
[3] Oracle, "Cipher (Java 8)," Oracle Java Documentation. [Online]. Available: https://docs.oracle.com/javase/8/docs/api/javax/crypto/Cipher.html
[4] Apache PDFBox. [Online]. Available: https://pdfbox.apache.org/
[5] Apache Commons Imaging. [Online]. Available: https://commons.apache.org/proper/commons-imaging/
[6] Apache POI. [Online]. Available: https://poi.apache.org/
[7] K. What, "JNativeHook Key Listener," GitHub. [Online]. Available: https://github.com/kwhat/jnativehook
[8] Instructor, "Technical Writing Style Guidelines," PPT, 2025.
[9] Instructor, "Objectives in Technical Report Writing," DOCX, 2025.

APPENDIX A: CLASS DIAGRAMS



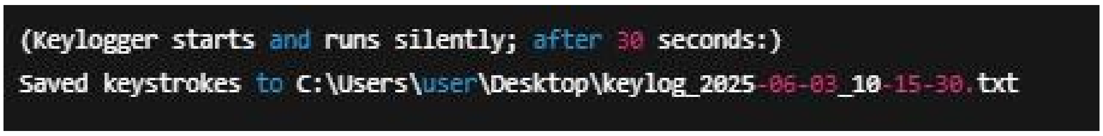
APPENDIX B: SAMPLE CONSOLE INTERACTION LOGS

I. File Integrity Checker



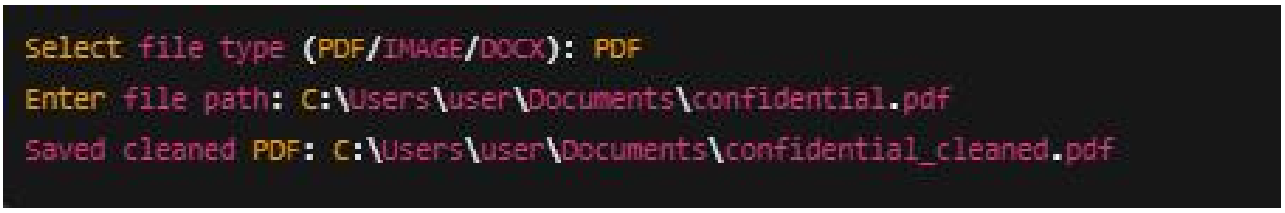
```
Welcome to the File Integrity Checker!
Type 'generate' to generate and save a checksum.
Type 'verify' to verify a file using saved checksum.
Enter your choice: generate
Enter the full path of the file: C:\Users\user\Documents\report.pdf
Enter the directory path where you want to save the checksum file: C:\Users\user\Documents\c
Checksum saved successfully.
```

II. Keylogger



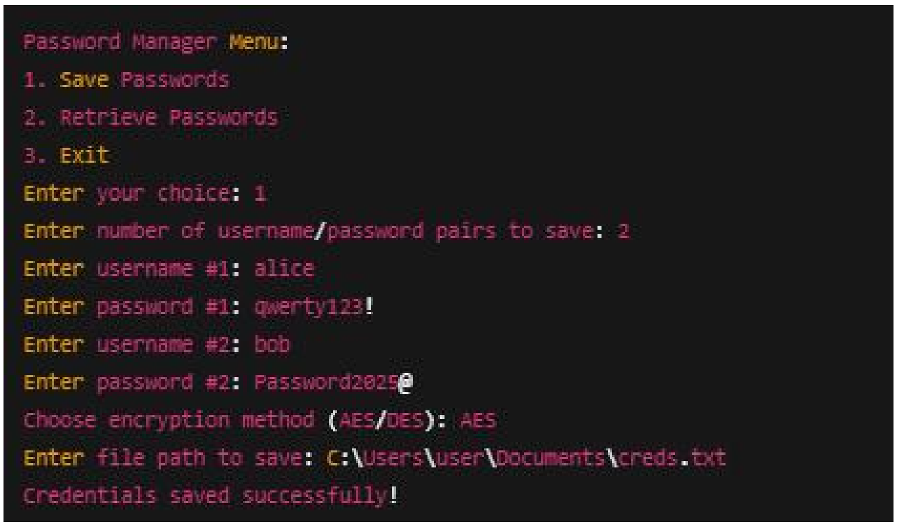
```
(Keylogger starts and runs silently; after 30 seconds:)
Saved keystrokes to C:\Users\user\Desktop\keylog_2025-06-03_10-15-30.txt
```

III. Metadata Remover



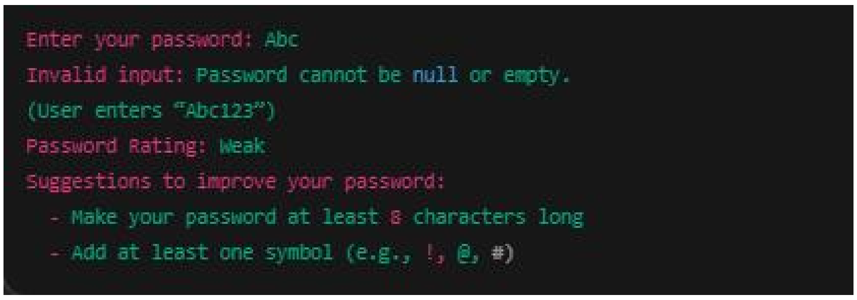
```
Select file type (PDF/IMAGE/DOCX): PDF
Enter file path: C:\Users\user\Documents\confidential.pdf
Saved cleaned PDF: C:\Users\user\Documents\confidential_cleaned.pdf
```

IV. Password Manager



```
Password Manager Menu:
1. Save Passwords
2. Retrieve Passwords
3. Exit
Enter your choice: 1
Enter number of username/password pairs to save: 2
Enter username #1: alice
Enter password #1: qwerty123!
Enter username #2: bob
Enter password #2: Password2025@
Choose encryption method (AES/DES): AES
Enter file path to save: C:\Users\user\Documents\creds.txt
Credentials saved successfully!
```

V. Password Strength Checker



```
Enter your password: Abc
Invalid input: Password cannot be null or empty.
(User enters "Abc123")
Password Rating: Weak
Suggestions to improve your password:
  - Make your password at least 8 characters long
  - Add at least one symbol (e.g., !, @, #)
```