Company Name: Shentinelix Sphere Pvt Ltd

Role: Cybersecurity Intern

Cryptographic Security Framework for File Integrity & Tamper Detection Project Report

Submitted By

Harnil Modi

Nov – Dec 2025

TABLE OF CONTENT

Abstract **3**
Chapter 1: Introduction **4**
1.1 Introduction 4
1.2 Background of File Integrity and Malware Threats 4
1.3 Need for File Integrity Verification 4
1.4 Problem Statement 4
1.5 Objectives of the Project 4
1.6 Scope of the Project 5
Chapter 2 — Literature Review **6**
2.1 Overview of Cryptographic Security 6
2.2 File Integrity Monitoring Techniques 6
2.3 Hashing Algorithms for Integrity Verification 7
2.4 Encryption Techniques for Data Protection 7
2.5 Cryptography in Malware and File Tampering Detection 8
2.6 Research Gap and Problem Identification 8
2.7 Summary 8
Chapter 3: System Methodology **9**
3.1 Requirement Analysis 9
Software Requirements 9
Functional Requirements 9
3.2 System Design Overview 10
Key Design Principles: 10
3.3 Cryptographic Framework Architecture 11
3.4 Encryption and Key Management Process 11
Process: 11
3.5 Hash Generation and Integrity Verification Process 12
Hash Generation: 12
Integrity Verification: 12
3.6 Logging and Alert Generation 12
Logging Details: 12
3.7 Summary 12
Chapter 4: Implementation and Results **13**
4.1 Implementation Environment 13
4.2 Encryption Module Implementation 13
Implementation Details: 13
4.3 Hash Generation Module Implementation 14
Implementation Details: 14
4.4 Integrity Verification Module Implementation 14

Implementation Details:..... 15
4.5 Logging Mechanism.....15
Logged Events:.....15
4.6 Testing Methodology.....16
Test Steps:..... 16
4.7 Results and Observations.....16
Results:.....16
Observations:.....16
4.8 Summary..... 16
Chapter 5 Diagram……..17
Chapter 6: Limitations and Future Enhancements.....18
6.1 Limitations.....18
6.2 Future Enhancements.....18
Chapter 7: Conclusion..... 19
7.1 Conclusion..... 19

Abstract

In modern cybersecurity environments, malware and unauthorized access often result in silent modification of critical files. Detecting such tampering is a key responsibility of security teams and Security Operations Centers (SOC).

This project implements a **Cryptographic Security Framework** that encrypts files and verifies their integrity using industry-standard cryptographic algorithms. The framework uses **AES-based encryption** to protect file confidentiality and **SHA-256 hashing** to detect unauthorized modifications.

If a file is altered, the system immediately raises an alert and logs the event, assisting security analysts in identifying potential malware activity. The project demonstrates practical cybersecurity concepts relevant to an internship role at Shentinelix Sphere Pvt Ltd.

Chapter 1: Introduction

1.1 Introduction

Cyber threats such as malware, ransomware, and insider attacks frequently target files to gain persistence or manipulate sensitive data

1.2 Background of File Integrity and Malware Threats

- Malware modifies files by injecting malicious code, altering configurations, or replacing legitimate data to maintain persistence. File integrity is critical in cybersecurity because even small unauthorized changes can compromise system security and trust.

1.3 Need for File Integrity Verification

Cryptographic techniques provide a reliable method for ensuring data confidentiality and integrity

1.4 Problem Statement

- Malware can modify files without triggering traditional alerts
  Detecting unauthorized file changes is critical during incident response
- Organizations need simple tools to verify file integrity
- There is a need for a beginner-friendly yet practical security solution

1.5 Objectives of the Project

- To design and develop a cryptographic security framework for file protection
- To encrypt sensitive files using secure cryptographic algorithms (AES)
- To generate cryptographic hashes (SHA-256) for file integrity verification

- To detect unauthorized file modifications caused by malware or malicious activity
- To generate alerts when file tampering is detected
- To log security-related events for investigation and auditing purposes
- To simulate a SOC-oriented file integrity verification use case

1.6 Scope of the Project

- The framework operates on user-selected files
- Supports file encryption and decryption
- Supports SHA-256 hash generation and integrity verification
  Designed to work on Windows operating systems
- Focused on detection of file tampering rather than prevention
- Uses a command-line interface (CLI) for execution
  Suitable for academic learning and cybersecurity internship-level implementation

Chapter 2 — Literature Review

2.1 Overview of Cryptographic Security

Cryptography is the science of protecting information using mathematical techniques, ensuring **confidentiality, integrity, and authentication** of data. It transforms plaintext into ciphertext so that only intended recipients can interpret it, preventing unauthorized access and tampering. Cryptographic methods form the foundation of modern digital security systems used in communications, data storage, and authentication protocols. GeeksforGeeks+1

Cryptographic techniques not only protect confidentiality but also ensure that data has not been altered, making them integral to secure systems. They include symmetric key algorithms (e.g., AES), asymmetric key algorithms (e.g., RSA), and hash functions (e.g., SHA-256). GeeksforGeeks+1

2.2 File Integrity Monitoring Techniques

Maintaining file integrity is critical for detecting unauthorized changes or malware tampering. Traditional error-detecting codes like checksums can detect accidental corruption but are insufficient for detecting deliberate tampering. Cryptographic hash functions such as SHA-256 provide a more robust method by generating a unique digest for a file, where any change — even a single bit — results in a different hash. Wikipedia+1

Cryptographic hash functions are widely used in security systems because they can reliably verify that a file has not been modified since the hash was generated. This property is essential for malware detection and file integrity assurance. Wikipedia

2.3 Hashing Algorithms for Integrity Verification

Hash functions are mathematical algorithms that map data of arbitrary size to fixed-length values called digests. A good cryptographic hash function is **one-way (non-invertible)** and produces a unique value for each unique input, making it extremely difficult for an attacker to forge a different file with the same hash. Wikipedia

SHA-256, part of the SHA-2 family, is widely used for integrity checks because it produces a 256-bit digest and has strong resistance against collision and pre-image attacks. This makes SHA-256 suitable for verifying file integrity in security systems and detecting tampering. Wikipedia

---

2.4 Encryption Techniques for Data Protection

Encryption converts readable data (plaintext) into unreadable ciphertext using algorithms and keys. It is a fundamental technique for protecting sensitive information from unauthorized access. There are two main types of encryption:

- **Symmetric Encryption:** Uses a single shared key for encryption and decryption (e.g., AES). It is efficient for encrypting large datasets.

- **Asymmetric Encryption:** Uses a pair of public and private keys, enabling secure key exchange and digital signatures. GeeksforGeeks+1

AES (Advanced Encryption Standard) is a symmetric encryption algorithm widely adopted due to its high performance and strong security, making it a

preferred choice for bulk data encryption in practical security applications. GeeksforGeeks

2.5 Cryptography in Malware and File Tampering Detection

Cryptographic techniques are also used to detect malicious modifications in files. By generating a secure hash before file use and then re-checking that hash later, systems can identify any unauthorized changes. This approach is particularly effective for malware detection, where attackers often modify system or configuration files to persist or hide malicious actions. ResearchGate+1

Research shows that using secure hashing algorithms like HMAC-SHA256 enhances file integrity verification by adding cryptographic authentication, which guards against tampering more effectively than simple hashing alone. ResearchGate

2.6 Research Gap and Problem Identification

Existing literature primarily focuses on cloud storage contexts or hybrid cryptographic models. However, many implementations are **complex or enterprise-oriented**, and there is a gap for **lightweight, easy-to-implement frameworks** designed for file integrity verification and malware tamper detection at the endpoint level. This project addresses that gap by creating a simplified framework suitable for cybersecurity learning and SOC-oriented workflows. IJERT

2.7 Summary

This literature review covered the foundational aspects of cryptography, hash-based integrity, encryption techniques, and their relevance to detecting unauthorized file modification. The review also identified the need for a practical, beginner-friendly framework — a need that this project aims to fulfill.

Chapter 3: System Methodology

This chapter explains the step-by-step methodology followed to design, implement, and validate the Cryptographic Security Framework for file integrity and tamper detection.

3.1 Requirement Analysis

The requirement analysis phase identifies the resources and tools needed to implement the system successfully.

Software Requirements

- Python 3.x
- Cryptography library
- Windows Operating System
- Visual Studio Code (VS Code)
- Windows PowerShell

Functional Requirements

- Ability to generate a secure encryption key
- Encrypt files securely
- Generate cryptographic hash values
- Verify file integrity
- Detect unauthorized file modification

- Log security-related events

Non-Functional Requirements

- Simple and user-friendly command-line interface
- Lightweight and fast execution
- Secure handling of cryptographic keys
- Reliability and accuracy in integrity detection

3.2 System Design Overview

The system is designed as a **command-line based cryptographic framework** with modular components. Each module performs a specific security function, ensuring clarity and maintainability.

Key Design Principles:

- Modular architecture
- Separation of encryption and integrity functions
- Secure key management
- Minimal system overhead

The framework allows users to perform security operations using predefined command-line arguments.

3.3 Cryptographic Framework Architecture

The architecture follows a linear and logical workflow:

1. User provides a file as input
2. Secure encryption key is generated
3. File is encrypted using AES
4. Hash value of the file is generated using SHA-256
5. File integrity is verified by comparing hashes

6. Alerts and logs are generated in case of tampering

This architecture ensures that both **confidentiality** and **integrity** of files are maintained.

3.4 Encryption and Key Management Process

Encryption is implemented using **AES-based symmetric encryption**.

Process:

- A secure encryption key is generated using a cryptographically strong random function
- The key is stored securely in a key file
- The original file is encrypted using the generated key
- The encrypted file is saved with a .enc extension

This process ensures that sensitive data remains unreadable to unauthorized users.

3.5 Hash Generation and Integrity Verification Process

Hashing is used to ensure file integrity.

Hash Generation:

- SHA-256 hashing algorithm is applied to the original file
- The generated hash is stored as a reference

Integrity Verification:

- Hash of the current file is recalculated
- The new hash is compared with the stored hash
- Any mismatch indicates file tampering

This method can detect even a single-byte modification in a file.

3.6 Logging and Alert Generation

To support investigation and auditing, the framework maintains security logs.

Logging Details:

- Key generation events
- File encryption and decryption activities
- Hash generation actions
- File tampering alerts

When unauthorized modification is detected, the system:

- Displays an alert message
- Records the incident in a log file

This logging mechanism supports SOC-style analysis and incident response.

3.7 Summary

This chapter described the complete methodology used to design and implement the cryptographic security framework. The approach focuses on secure encryption, reliable integrity verification, and effective logging to detect file tampering. The methodology ensures the framework is practical, lightweight, and suitable for cybersecurity internship-level learning and SOC-oriented workflows.

Chapter 4: Implementation and Results

This chapter explains the implementation details of the Cryptographic Security Framework along with the testing process and results obtained during execution.

4.1 Implementation Environment

The project was implemented and tested in the following environment:

- **Operating System:** Windows Operating System
- **Programming Language:** Python 3
- **Development Tool:** Visual Studio Code (VS Code)
- **Execution Interface:** Windows PowerShell
- **Libraries Used:** Python Cryptography Library

The environment was selected to simulate a real-world endpoint system commonly used in organizational and SOC environments.

4.2 Encryption Module Implementation

The encryption module is responsible for protecting file confidentiality.

Implementation Details:

- A secure symmetric encryption algorithm (AES) is used
- An encryption key is generated using a cryptographically secure method
- The key is stored in a separate key file

- The input file is encrypted using the generated key
- The encrypted output is saved with a .enc file extension

This module ensures that sensitive files cannot be read without proper authorization.

4.3 Hash Generation Module Implementation

The hash generation module is used to create a baseline for file integrity verification.

Implementation Details:

- The SHA-256 hashing algorithm is applied to the file
- The generated hash is unique for the file content
- The hash value is stored in a separate file for later comparison

SHA-256 ensures that even minor changes in file content produce a completely different hash value.

4.4 Integrity Verification Module Implementation

The integrity verification module detects unauthorized file modifications.

Implementation Details:

- The hash of the current file is recalculated
- The newly generated hash is compared with the stored original hash
- If both hashes match, file integrity is confirmed
- If hashes differ, file tampering is detected

This mechanism helps identify malware-driven or unauthorized file changes.

4.5 Logging Mechanism

A logging mechanism is implemented to record all security-related activities.

Logged Events:

- Encryption key generation
- File encryption and decryption
- Hash generation
- File integrity verification
- Tampering detection alerts

The logs are stored in a log file and can be used for auditing, troubleshooting, and SOC-style investigations.

4.6 Testing Methodology

Testing was performed to validate the functionality and reliability of the system.

Test Steps:

1. A sample text file was created
2. A cryptographic hash was generated for the file
3. The file was modified to simulate malware behavior
4. Integrity verification was performed

The testing approach ensured both normal and malicious scenarios were evaluated.

4.7 Results and Observations

Results:

- The framework successfully encrypted files
- Hash values were generated correctly
- Unauthorized file modification was detected accurately
- Alert messages were displayed upon tampering
- Security events were logged successfully

Observations:

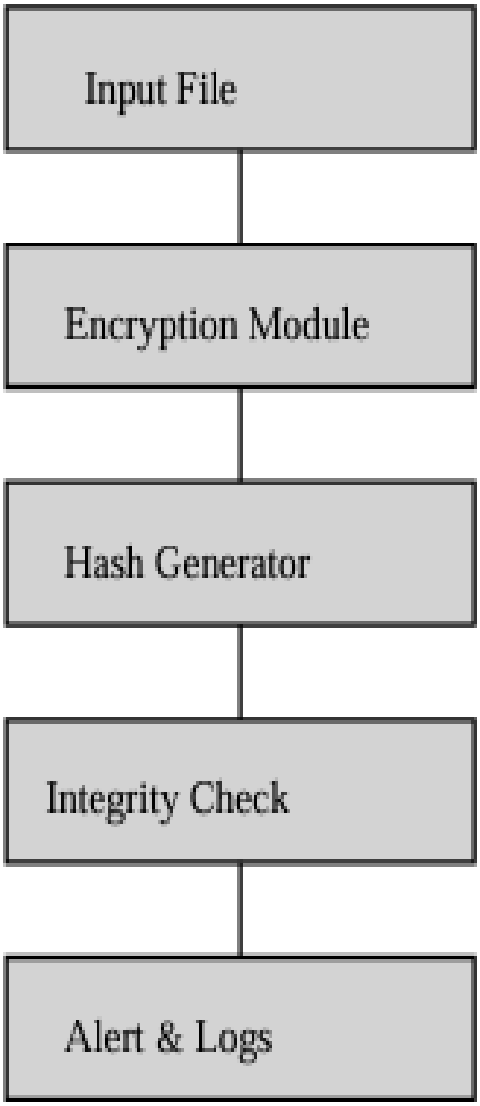
- The system reliably detected file changes
- Even small modifications were identified
- The framework performed efficiently with minimal overhead

These results confirm that the framework functions as intended.

4.8 Summary

This chapter presented the implementation details and results of the Cryptographic Security Framework. The system successfully demonstrated secure file encryption, reliable hash-based integrity verification, and effective tamper detection. The results validate the framework's applicability for cybersecurity internship-level learning and SOC-oriented file integrity monitoring.

Chapter 5: Diagram



Chapter 6: Limitations and Future Enhancements

6.1 Limitations

The limitations of the proposed system are:

- The framework requires manual execution through the command-line interface
- Real-time file monitoring is not implemented
- The system does not include direct SIEM integration
- Only file-level detection is supported
- Key management is basic and local to the system

6.2 Future Enhancements

The system can be enhanced in the future by:

- Implementing real-time file integrity monitoring
- Integrating the framework with SIEM tools such as Splunk or Wazuh
- Adding ransomware behavior detection techniques
- Developing a graphical user interface (GUI)
- Automating alert forwarding to security teams

Chapter 7: Conclusion

7.1 Conclusion

The Cryptographic Security Framework for File Integrity and Tamper Detection successfully demonstrates the use of cryptographic techniques to protect files and detect unauthorized modifications. By combining AES-based encryption with SHA-256 hashing, the system ensures confidentiality and integrity of data.

The project provides practical exposure to cryptography, secure coding, and security monitoring concepts relevant to SOC and incident response environments. This framework serves as a strong foundation for cybersecurity internship-level learning and can be extended for advanced security applications.