Department of Telecom and Networking

Course: CryptoGraphy

Term 1 | Year 3

Name: Kunthea Sochetra

Project Report:

Title: Anti Ransomware monitoring System

# I. Project Overview

### 1 Problem Statement

Ransomware attacks encrypt files silently, often going undetected until ransom demands appear. Traditional antivirus solutions frequently identify threats too late, after significant data loss has occurred. There is a critical need for proactive systems that can detect file tampering in real-time.

### 2 Solution Approach

This system implements a cryptographic defense mechanism:

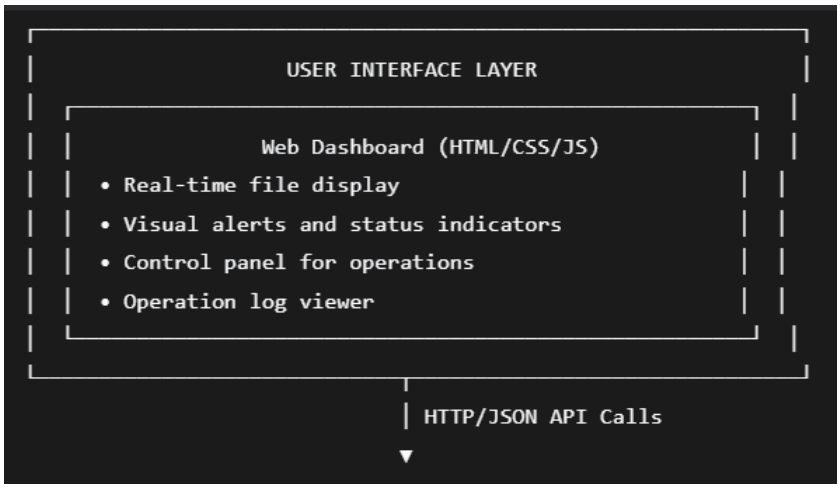
- Creates SHA-256 hash "fingerprints" of all files
- Continuously monitors file integrity (every 2 seconds)
- Compares current hashes with original baselines
- Immediately alerts on any unauthorized changes
- Provides a web dashboard for real-time monitoring

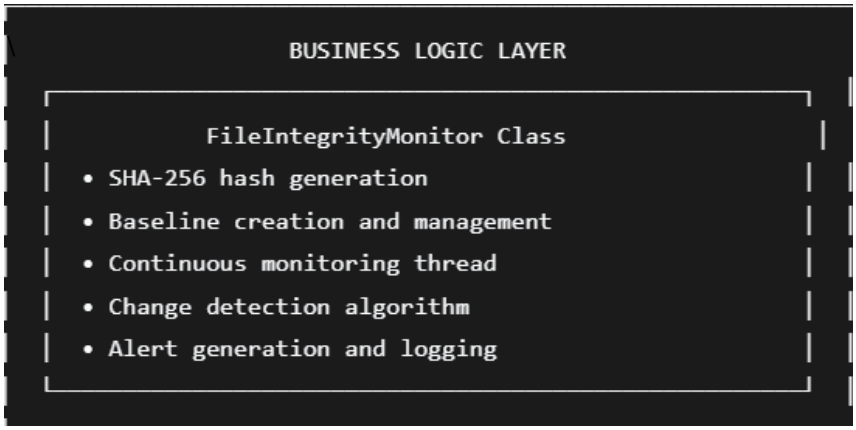
### 3 Key Features

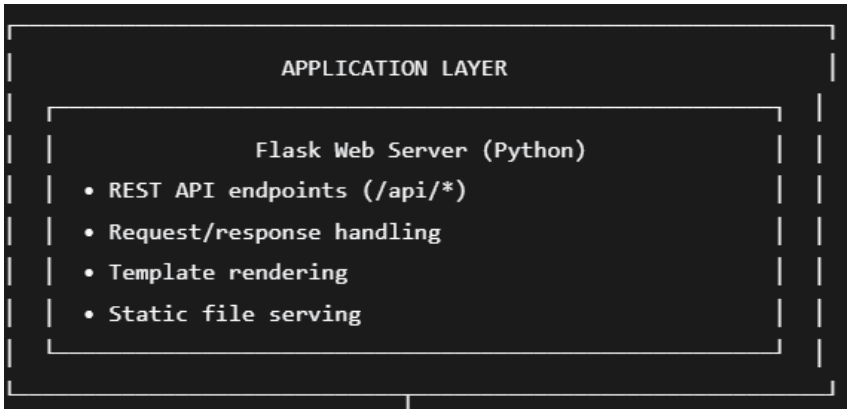
- Real-time file integrity monitoring
- SHA-256 cryptographic hashing
- Web-based dashboard interface
- Configurable scan intervals (1-60 seconds)
- Detailed operation logging
- Immediate visual alerts
- File status categorization (unchanged/modified/new/deleted)
- Performance statistics tracking

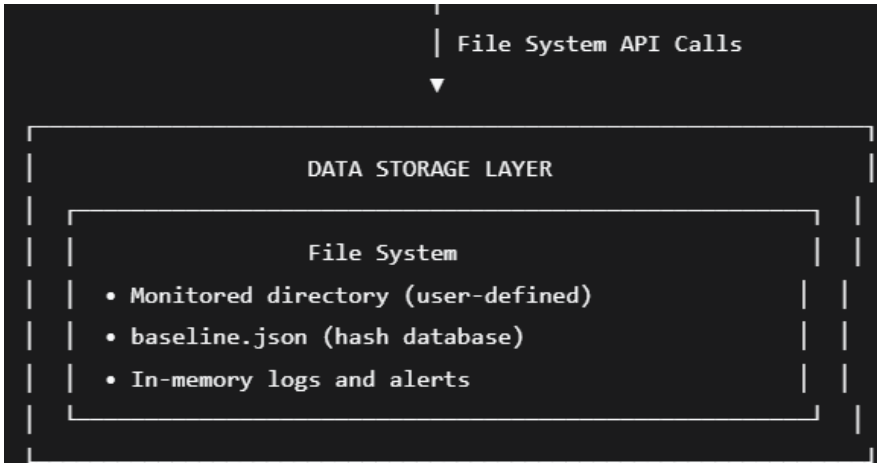
## II. System Design / Architecture

### 1. System Architecture

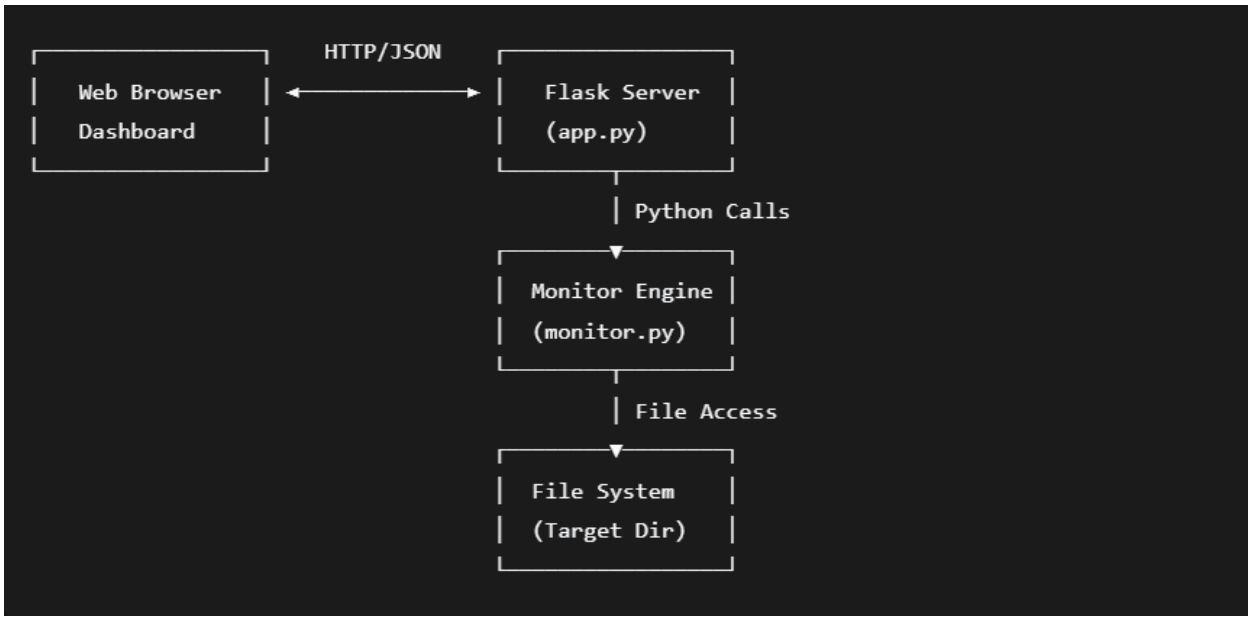





## 2 Component Structure



# III. Implementation Details

## Key Components

### 1. Core Monitoring Engine (core/monitor.py)

```python
class FileIntegrityMonitor:
    def __init__(self):
        # Core attributes
        self.directory = None          # Path to monitor
        self.baseline = {}             # {filepath: {hash, size, modified}}
        self.alerts = []               # Alert history
        self.stats = {}                # Monitoring statistics
        self.is_monitoring = False     # Monitoring state
        self.operation_logs = []       # Detailed operation logs
```

### 2. Cryptographic Hash Generation

```python
def generate_hash(self, filepath):
    """Generate SHA-256 hash for a file"""
    sha256_hash = hashlib.sha256()
    with open(filepath, "rb") as f:
        # Read in 8KB chunks for memory efficiency
        for byte_block in iter(lambda: f.read(8192), b""):
            sha256_hash.update(byte_block)
    return sha256_hash.hexdigest()  # Returns 64-character hex string
```

### 3. Baseline Creation Algorithm

```
Algorithm: CREATE_BASELINE
Input: directory_path
Output: baseline dictionary

1. Initialize empty baseline dictionary
2. FOR each file in directory (recursive):
   a. Generate SHA-256 hash
   b. Extract file metadata (size, modified time)
   c. Store: baseline[filepath] = {hash, metadata}
   d. Log progress every 100 files
3. Save baseline to JSON file
4. Return success status
```

## 4. File Comparison Algorithm

```
Algorithm: COMPARE_FILES
Input: current_files, baseline
Output: change detection results

1. Initialize statistics counters
2. FOR each file in current directory:
   a. IF file not in baseline:
        - Mark as NEW file
        - Generate alert
   b. ELSE:
        - Generate current hash
        - IF current_hash ≠ baseline_hash:
            * Mark as MODIFIED
            * Generate RANSOMWARE ALERT
        - ELSE:
            * Mark as UNCHANGED
3. FOR each file in baseline not in current:
   a. Mark as DELETED
   b. Generate alert
```

## 5. Web Server (app.py)

```
# REST API Endpoints
@app.route('/api/set-directory', methods=['POST'])      # Set monitoring directory
@app.route('/api/create-baseline', methods=['POST'])    # Create hash baseline
@app.route('/api/scan', methods=['POST'])               # Single scan
@app.route('/api/start-monitoring', methods=['POST'])   # Start continuous monitoring
@app.route('/api/stop-monitoring', methods=['POST'])    # Stop monitoring
@app.route('/api/data')                                 # Get current status
@app.route('/api/logs')                                 # Get operation logs
@app.route('/api/scan-history')                         # Get scan performance
```

## Data Structure

### 1. Baseline Structure

```
baseline = {
    "/path/to/file1.txt": {
        "hash": "a1b2c3d4e5f6...",  # SHA-256 hash
        "size": 1024,               # File size in bytes
        "modified": 1634567890.123, # Last modified timestamp
        "path": "/path/to/file1.txt",
        "name": "file1.txt"
    },
    # ... more files
}
```

### 2. Alert Structure

```
alert = {
    "type": "danger",               # success/warning/danger/info
    "message": "FILE MODIFIED...",
    "filepath": "/path/to/file.txt",
    "timestamp": "2024-01-15 14:30:25.456",
    "id": 1642257025456             # Unique identifier
}
```

### 3. File Status Structure

```
file_info = {
    "name": "document.txt",
    "path": "/home/user/documents",
    "full_path": "/home/user/documents/document.txt",
    "status": "modified",          # unchanged/modified/new/deleted
    "hash": "a1b2c3d4...",        # Truncated hash for display
    "size": 2048                  # File size in bytes
}
```

## Libraries and Dependencies

### Core Libraries:

- Flask (2.3.3): Web framework for dashboard and API
- hashlib: Python's built-in cryptographic hash functions
- pathlib: Object-oriented filesystem paths
- threading: Concurrent monitoring execution
- json: Baseline storage serialization

### Cryptographic implementation

```
import hashlib
# SHA-256 provides:
# - 256-bit output (64 hex characters)
# - Collision resistance
# - Preimage resistance
# - Avalanche effect (small changes → completely different hash)
```

## IV. Usage Guide

### Installation Steps

## 1. Prerequisites

```
# Python 3.8 or higher
python --version

# Required system tools
# - Windows: PowerShell 5.1+
# - Linux/macOS: bash terminal
```

## 2. Setup Project Structure

```
# Create project directory
mkdir anti-ransomware-monitor
cd anti-ransomware-monitor

# Create directory structure
mkdir static\css
mkdir templates
mkdir core

# Create empty baseline file
echo {} > baseline.json

# Create core package init file
echo # Empty file > core\__init__.py
```

## 3. Install Dependencies

```
# Create requirements.txt
echo Flask==2.3.3 > requirements.txt

# Install using pip
pip install -r requirements.txt
```

### 4. Copy Source Files

**Copy the following files to their respective locations:**

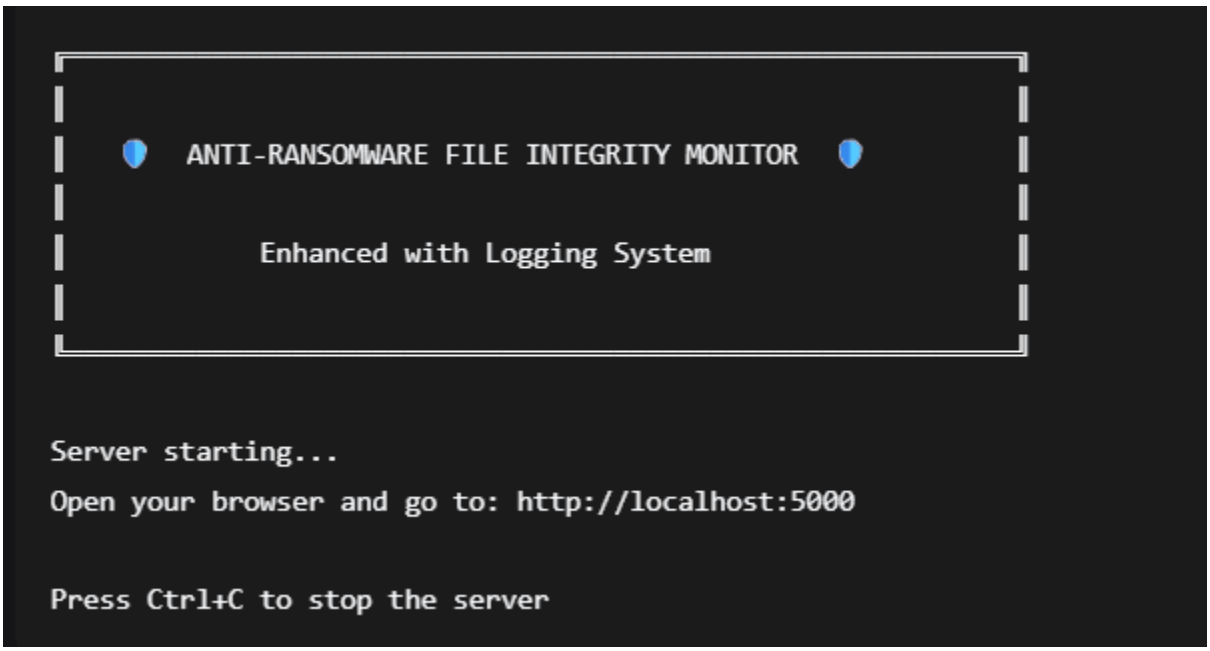
- app.py → Root directory
- core/monitor.py → core/ directory
- templates/index.html → templates/ directory
- static/css/style.css → static/css/ directory

**#Running the Application**

### 1. Start the Server

```
python app.py
```

**Expected Output:**



- **Access the Dashboard**

1. Open web browser
2. Navigate to: http://localhost:5000
3. You should see the Anti-Ransomware Monitor dashboard

# Step-by-Step Usage

### Step 1: Set Monitoring Directory

1. Enter directory path in "Directory to Monitor" field
    - Example: ./test_files (relative path)
    - Example: C:\Users\YourName\Documents (absolute path)
2. Click "Set Directory" button
3. Verify success message appears

### Step 2: Create Baseline

1. Click "Create Baseline" button
2. Wait for processing (progress shown in logs)
3. Check baseline.json is created with file hashes
4. Verify success alert appears

### Step 3: Test Single Scan

1. Click "Single Scan" button
2. Observe:
    - File list updates
    - Status badges show file states
    - Statistics update
    - Logs show scan details

**Step 4: Start Continuous Monitoring**

1. Set scan interval (default: 2 seconds)
2. Click "Start Monitoring" button
3. Verify:
    - Monitoring status changes to "Active"
    - Stop button appears
    - Logs show monitoring started
    - Automatic scans begin

**Step5: Ransomware detection**

1. Create a test file in monitored directory:

```
echo "Original content" > test.txt
```

2.Modify the file (simulate ransomware):

```
echo "ENCRYPTED_CONTENT" > test.txt
```

Observe detection:

- Red "MODIFIED" alert appears
- Statistics show modified files count increases
- Alert list shows ransomware warning
- Logs capture the event

## Troubleshooting

### Common Issues and Solutions:

1. "Directory does not exist" error
    - Verify path spelling
    - Use forward slashes or double backslashes on Windows
    - Ensure directory permissions allow read access
2. No alerts when files change
    - Ensure files are saved (text editors may buffer changes)
    - Check scan interval is not too long
    - Verify baseline was created successfully
3. Web page not loading
    - Check Flask server is running
    - Verify port 5000 is not blocked
    - Clear browser cache (Ctrl+F5)
4. Slow performance with many files
    - Increase scan interval
    - Exclude unnecessary file types
    - Monitor smaller directories

**Dashboard Indicators:**

- Green: All files secure (unchanged)
- Yellow: New files detected (investigation needed)
- Red: Modified/deleted files (immediate action required)

## V. Conclusion and Future Work

Conclusion

The Anti-Ransomware File Integrity Monitor successfully demonstrates:

1. Practical Cryptography Application: SHA-256 hashing for file integrity
2. Real-time Detection: Continuous monitoring with 2-second intervals
3. User-Friendly Interface: Web dashboard with visual alerts
4. Comprehensive Logging: Detailed operation tracking
5. Educational Value: Clear demonstration of cryptographic principles

Key Achievements:

- Successfully detects file modifications in real-time
- Provides immediate visual and log-based alerts
- Maintains detailed history of file changes
- Offers intuitive control interface
- Demonstrates practical application of hash functions

Future Enhancements

1. Enhanced Cryptographic Features

- Digital Signatures: Add RSA signatures to baseline for authenticity
- HMAC Integration: Keyed-hash message authentication for baseline
- Bloom Filters: Efficient file existence checking for large directories
- Incremental Hashing: Update hashes without full rescan

2. Advanced Monitoring Capabilities

```
# Proposed enhancement: File type-specific monitoring
monitoring_profiles = {
    "documents": [".doc", ".pdf", ".txt"],
    "images": [".jpg", ".png", ".gif"],
    "databases": [".db", ".sql", ".mdb"]
}

# Proposed: Behavior-based detection
def detect_suspicious_pattern(file_changes):
    # Multiple rapid modifications
    # Encryption-like file size changes
    # Suspicious file extensions
    # Known ransomware patterns
```

3. **System Architecture Improvements**

- Distributed Monitoring: Monitor multiple directories/servers
- Centralized Dashboard: Aggregate alerts from multiple instances
- Database Backend: Replace JSON files with SQL database
- Cloud Integration: AWS S3/Google Drive monitoring

4. **User Experience Enhancements**

- Mobile Application: iOS/Android monitoring apps
- Email/SMS Alerts: Notifications for critical changes
- Historical Analysis: Trend graphs and reports
- Automated Responses: Quarantine modified files

5. **Security Hardening**

- Encrypted Baseline: AES encryption for baseline.json
- Authentication: User login system
- Audit Trail: Comprehensive activity logging
- Backup Integration: Automatic restoration from backups

6. Performance Optimization

- Multithreading: Parallel file hashing
- GPU Acceleration: CUDA-accelerated hash calculations
- Delta Checking: Only hash changed portions of files
- Caching System: Store frequently accessed file hashes

## VI. References

Academic References


1. Stallings, W. (2017). Cryptography and Network Security: Principles and Practice. Pearson.
    - Chapter 11: Cryptographic Hash Functions
    - SHA-256 algorithm details and applications

2. Menezes, A. J., van Oorschot, P. C., & Vanstone, S. A. (2018). Handbook of Applied Cryptography. CRC Press.
    - Hash function design principles
    - Digital fingerprinting techniques

3. Paar, C., & Pelzl, J. (2010). Understanding Cryptography: A Textbook for Students and Practitioners. Springer.
    - Practical cryptographic implementations
    - Security protocol design

Technical Documentation

1. NIST FIPS 180-4 (2015). Secure Hash Standard (SHS)
    - Official SHA-256 specification
    - Implementation guidelines

2. Python Documentation
    - hashlib library: https://docs.python.org/3/library/hashlib.html
    - Flask framework: https://flask.palletsprojects.com/

3. OWASP Cheat Sheet Series
    - File Integrity Monitoring: https://cheatsheetseries.owasp.org/