FILE INTEGRITY CHECKER

<Add>

Table of Contents

Table of Figures

Index of Tables

1. Abstract

This report examines the development and functionality of the File Integrity Checker designed specifically for Microsoft Office files. The tool serves as a safeguard against unauthorized modifications to essential data, programs, configurations, and security information stored within computer file systems. By comparing Microsoft Office files to a predefined database, the File Integrity Checker aims to ensure their authenticity and detect any unauthorized changes. Utilizing cryptographic algorithms, the tool generates unique hash codes for comparison, providing users with a reliable method to verify file integrity. Emphasizing usability and flexibility, the File Integrity Checker caters to users of varying technical proficiency, promoting broader adoption of file integrity practices. This report explores the significance of file integrity in the context of data security and highlights the role of the File Integrity Checker in mitigating risks associated with unauthorized file alterations.

2. Introduction

In today’s digital landscape ensuring integrity of files is one of the most important aspects of data security. As what we store and share rapidly increases the risk of unauthorized modification of our data is turning out to be a major concern. We need to assure that our files remain non modified to trust the authenticity of digital assets.

The hash code generator for Microsoft applications uses cryptographic algorithms to create a unique hash code which in turn ensures that even a small edit in the content will change the generated hash code. This makes it possible for users to determine the integrity of the file during initial phase and then compare with the recalculated hash code to check for any differences.

The File Integrity Checker for Microsoft Applications focuses on usability and flexibility, making it a practical and effective solution. This tool is intended for users with different technical skill levels. By providing user-friendly interfaces and flexible features, the File Integrity Checker empowers individuals with varying technical proficiencies to navigate and leverage its capabilities effectively. This emphasis on usability not only enhances the accessibility of the tool but also promotes a broader adoption of file integrity practices among users with diverse skill sets. In an era where cybersecurity should be accessible to all, the File Integrity Checker stands as a testament to the commitment to user inclusivity and data security.

3. Literature review

The digital realm, once a mere frontier, has become the bedrock of modern society, underpinning critical infrastructure, communication channels, and innumerable business operations. Organizations across sectors store and manage vast troves of data, encompassing sensitive financial records, intellectual property, and personal information. Maintaining the integrity and authenticity of this data is a imperative, ensuring trust, safeguarding privacy, and enabling reliable decision-making.

However, the digital landscape is a breeding ground for cyber threats, where malicious actors employ a diverse arsenal of techniques to compromise systems and manipulate data. Consequences of data breaches extend far beyond financial losses, potentially causing reputational damage, disrupting critical operations, and eroding user trust.

In this context, file integrity checking (FIC) and hash code generation have become indispensable tools for safeguarding digital assets. FIC ensures the accuracy and consistency of data by detecting unauthorized modifications to files, while hash codes, derived from cryptographic hash functions, act as unique digital fingerprints, enabling the identification of even minor changes. This literature review explores the theoretical underpinnings and practical applications of these techniques, highlighting their vital role in securing digital environments.

3.1 The Importance of File Integrity Checking

FIC safeguards data by detecting unauthorized alterations to files, whether accidental or malicious. Accidental changes can stem from hardware malfunctions or software bugs, while malicious modifications can be perpetrated by cybercriminals seeking to disrupt operations, steal sensitive information, or hold data hostage for ransom.

FIC plays a crucial role in upholding the CIA triad (Confidentiality, Integrity, and Availability) of information security by focusing on the integrity aspect. It contributes to:

- Protection against cyberattacks: Ransomware and other malicious software often target critical files. FIC acts as a defense mechanism, monitoring files and their hash values, providing early warnings for implementing recovery measures (Salman et al., 2022).

- Regulatory compliance: Industries with strict data security standards rely on FIC to ensure data reliability. It helps organizations meet regulatory requirements and prevent data tampering (Salman et al., 2022). For example, the Health Insurance Portability and Accountability Act (HIPAA) in the United States mandates the implementation of safeguards to ensure the integrity of protected health information (Department of Health and Human Services (HHS), 2003).
- Preventing unauthorized access: FIC helps identify and address attempts to gain unauthorized access to sensitive data by regularly monitoring and verifying file

integrity. This becomes crucial in sectors like finance and healthcare, where unauthorized access to data can have severe consequences.

- Maintaining system reliability: A reliable system hinges on the integrity of its system files. FIC identifies changes that might impact system stability and functionality, enabling proactive maintenance (Gene & Eugene, 1994). For instance, altered system configuration files can lead to malfunctions, highlighting the importance of FIC for system stability.

- Ensuring data accuracy: Data accuracy is paramount for various sectors, especially those dealing with critical information like financial records, scientific data, and medical reports. FIC helps detect and rectify discrepancies or alterations in stored files, ensuring data accuracy for decision-making and analyses.

Beyond these core benefits, FIC facilitates:

- Timely incident response: Early detection of changes in file integrity allows organizations to swiftly initiate investigations and corrective actions, minimizing the potential damage caused by security incidents (Chen et al., 2017).

- Protection against insider threats: FIC safeguards against internal attempts to manipulate or compromise files. Regular checks help identify and mitigate such threats, which can be especially detrimental due to insider access privileges.

- Auditing and accountability: FIC provides an audit trail for digital assets, enhancing accountability. By maintaining a record of file changes, organizations can trace the history of modifications for forensic analysis and assessments in case of security breaches or legal disputes.

- Enhancing trustworthiness: Trust is essential in digital environments, especially for e-commerce and online transactions. FIC promotes user, client, and stakeholder trust by ensuring data remains unaltered and reliable. This fosters confidence in the integrity of online transactions and interactions.

FIC tools come in various forms, each offering advantages and disadvantages depending on the specific needs. Common types include:

1. Tripwire software: Monitors system files and directories, recording their attributes over time. Any discrepancy between the stored and calculated attributes raises a red flag, indicating potential unauthorized modifications. Tripwire offers a centralized view of file system integrity, simplifying management and monitoring. However, it

can be resource-intensive for large systems and might generate false positives due to legitimate changes in file attributes.

2. File Integrity Monitoring Agents (FIMAs): These software agents reside on individual systems, continuously monitoring file integrity. They offer real-time monitoring capabilities and can be configured to trigger alerts for suspicious modifications. FIMAs are well-suited for distributed systems but require deployment and management on each device.

3. Anti-virus/Anti-malware Software: While primarily focused on detecting and removing malware, some antivirus solutions also incorporate basic FIC functionalities. They can scan files for known malicious code and flag unauthorized modifications associated with malware activity. However, their focus is primarily on malware detection, and their FIC capabilities might be limited compared to dedicated tools.

4. Operating System Features: Modern operating systems often include built-in integrity checking features. These might involve file system journaling, which tracks changes to files, or volume shadow copies, which create periodic snapshots of file systems for rollback purposes. While convenient, these built-in features might not offer the same level of granularity and control as dedicated FIC tools.

The choice of FIC tool depends on factors like:

1. Security requirements: Organizations with high-security needs might opt for robust tools like Tripwire, while others might prioritize ease of use and choose built-in OS features.
2. System size and complexity: Large and distributed systems might benefit from scalable solutions like FIMAs, while smaller systems might find centralized tools like Tripwire sufficient.
3. Technical expertise: Implementing and managing dedicated FIC tools requires technical expertise, while built-in OS features might be easier to use for less technical users.

The choice of FIC technique depends on factors such as:

- Security requirements: Organizations with high-security needs might opt for a combination of hashing and digital signatures, while others might prioritize efficiency with basic file hashing.

- File size and type: Hashing large files can be time-consuming. Techniques like file attribute monitoring can be a more efficient option for large datasets.

- Performance impact: FIC techniques can impact system performance. Careful consideration is needed to balance security needs with resource constraints.

Benefits and Limitations of FIC While FIC offers significant benefits for data integrity, it's crucial to acknowledge its limitations:

- False positives: FIC techniques might flag legitimate changes as suspicious, leading to unnecessary investigation and wasted resources.
- Limited scope: FIC primarily focuses on detecting changes, not preventing them. Additional security measures like access controls and intrusion detection systems are necessary for comprehensive protection.

- Hash function vulnerabilities: Older hash functions like MD5 are susceptible to collision attacks, where attackers can potentially generate different files with the same hash code. Using strong hash functions like SHA-256 mitigates this risk (Wang et al., 2005).

- Storage and management overhead: Securely storing and managing hash values for a large number of files can be challenging, especially in large-scale systems (Nicholson et al., 2003).

3.2 Hash Codes: The Guardians of File Integrity

Hash codes play a pivotal role in FIC. These are fixed-size character strings derived from cryptographic hash functions. These functions operate like mathematical filters, transforming the entire content of a digital file (the input) into a unique hash code (the output). The key characteristic of a cryptographic hash function is that any alteration, however minor, to the original file results in a completely different hash code.

Common hash function algorithms employed in FIC include:

1. MD5 (Message-Digest Algorithm 5): An older but widely used hash function with a 128-bit output. While still used in some legacy systems, MD5 is considered cryptographically broken due to its vulnerability to collision attacks (Easttom, 2014; Wang et al., 2005).
2. SHA-1 (Secure Hash Algorithm 1): A 160-bit hash function offering stronger collision resistance than MD5. However, recent advancements in cryptanalysis have raised concerns about its future security (National Institute of Standards and Technology (NIST), 2017).

1. SHA-256 (Secure Hash Algorithm 2): A secure and widely used hash function with a 256-bit output. SHA-256 offers a significantly higher level of collision resistance compared to MD5 and SHA-1 and is recommended by security experts for most applications (National Institute of Standards and Technology (NIST), 2017; Dang, 2012).

Here's how hash codes work with FIC:

- ***Initial Hashing:*** A cryptographic hash function is applied to the original file, generating a unique hash code. This hash code is then securely stored, often in a separate database or file.
- ***Regular Checks:*** Periodically or upon specific events (e.g., file access, system restart), the hash code is recalculated for the file.
- ***Comparison:*** The newly generated hash code is compared to the stored original hash code.
- ***Detection:*** If the hash codes don't match, it signifies a potential modification to the file, requiring further investigation.

This simple yet powerful mechanism ensures data integrity and allows for swift detection of unauthorized changes. Even a single byte alteration in the file results in a different hash code, effectively acting as a digital fingerprint for the file.

However, it's important to understand the limitations of hash codes:

- ***Collision Attacks*:** While highly improbable, it's theoretically possible for attackers to generate two different files with the same hash code using advanced mathematical techniques. This highlights the importance of using robust hash functions like SHA-256 (Wang et al., 2005).
- ***Pre-image Attacks*:** These attacks aim to find the original file content given a specific hash code. While computationally expensive, such attacks pose a risk for highly sensitive data. Regularly updating hash functions and using strong algorithms mitigates this risk (Hellman, 1980).
- ***Second Pre-image Attacks***: These attacks aim to create a different file with the same hash code as a specific existing file. This scenario is even more challenging than pre-image attacks and poses a lower threat for most applications (Rogaway & Shrimpton, 2004).

3.3 Applications of FIC and Hash Codes FIC and hash codes:

1. ***Information Security*:** Organizations leverage FIC and hash codes to safeguard critical data from unauthorized modifications. This is crucial in sectors like finance, healthcare, and government, where data integrity is paramount (Bayne et al., 2010).
2. ***Software Development*:** Software developers use FIC to verify the integrity of downloaded software packages, ensuring they haven't been tampered with during transmission. Hash codes are often published by software vendors to allow users to verify the authenticity of downloaded files (Guri et al., 2018).
3. ***Digital Forensics*:** In the aftermath of a cyberattack or security incident, FIC and hash codes play a vital role in forensic investigations. By analyzing file timestamps, modifications, and hash code discrepancies, investigators can reconstruct the timeline of events and identify compromised files (Kahvedzic & Kont, 2015).
4. ***Data Archiving*:** Long-term data storage often involves archiving for future reference. FIC and hash codes ensure the integrity of archived data over time, preventing accidental or malicious alterations (Rosenthal et al., 2005).
5. *Software Distribution*: Software vendors employ FIC and hash codes to ensure the authenticity and integrity of downloadable software updates. This protects users from installing tampered updates that could compromise their systems (Guri et al., 2018).

These are just a few examples, and the applications of FIC and hash codes continue to evolve as the digital landscape expands.

3.4 Blockchain Integration for Enhanced Trust and Transparency

As the digital landscape continues to evolve, the integration of FIC and hash codes with emerging technologies like blockchain has garnered significant attention. Blockchain, with its decentralized, immutable, and transparent nature, presents exciting opportunities for enhancing the trustworthiness and integrity of digital assets.

3.4.1 Blockchain for Data Integrity Assurance

Blockchain technology offers a novel approach to ensuring data integrity by leveraging its distributed ledger architecture and cryptographic principles. By recording file hash codes on the blockchain, a tamper-evident and auditable trail of data modifications can be established (Zheng et al., 2018). This approach can be particularly valuable in scenarios where multiple parties need to collaborate and maintain a shared, immutable record of data changes.

One key advantage of blockchain-based FIC is the elimination of a single point of failure or trust. Traditional centralized systems rely on a trusted third party to maintain the integrity of data, which can introduce vulnerabilities and potential points of compromise. With blockchain, the integrity verification process is decentralized, reducing the risk of data tampering or manipulation by any single entity (Casino et al., 2019).

Furthermore, the transparency inherent in blockchain systems enables all participants to verify and audit the integrity of data independently. This level of transparency can foster trust among stakeholders, particularly in scenarios where data integrity is critical, such as supply chain management, electronic voting, and medical record keeping (Huang et al., 2020).

3.4.2 Challenges and Considerations

While the integration of FIC and hash codes with blockchain technology holds great promise, several challenges and considerations must be addressed:

- Scalability: As blockchain networks grow in size and transaction volume, scalability issues may arise, potentially impacting the efficiency and performance of FIC operations (Croman et al., 2016). Research efforts are underway to develop scalable blockchain solutions that can handle large-scale FIC implementations.
- Privacy and confidentiality: While blockchain provides transparency, there may be scenarios where data confidentiality is a concern. Techniques such as zero-knowledge proofs and secure multi-party computation can be explored to enable privacy-preserving FIC on blockchain (Kosba et al., 2016).
- Interoperability: With multiple blockchain platforms and protocols in existence, ensuring interoperability and seamless integration with existing FIC systems is a crucial consideration (Paik et al., 2019).
- Regulatory and legal implications: The adoption of blockchain-based FIC solutions may require addressing legal and regulatory challenges, particularly in sectors with stringent data protection and compliance requirements (Yeoh, 2017).

Despite these challenges, the integration of FIC and hash codes with blockchain technology presents a promising avenue for enhancing data integrity, transparency, and trust in digital environments.

3.5 Future Directions in FIC and Hash Codes

The cyber threat landscape is constantly evolving, necessitating ongoing advancements in FIC and hash code technology. Here are some key areas for future research:

1. Self-healing file systems: These systems can automatically detect and repair inconsistencies in file integrity, potentially reducing reliance on manual intervention (Sundararaman et al., 2017).
2. Enhanced Intrusion Detection Systems (IDS): Integrating FIC data with IDS can provide a more comprehensive view of system activity, enabling more effective anomaly detection and faster response times (Zhou et al., 2010).
3. Lightweight and scalable FIC techniques: Developing more efficient and scalable FIC techniques is crucial for handling massive datasets and minimizing performance impact on large-scale systems (Aguilera et al., 2003).

4. Post-quantum cryptography: With the potential emergence of quantum computers, new cryptographic hash functions resistant to quantum attacks are being explored to ensure long-term security (Bernstein & Lange, 2017).
5. Integration with emerging technologies: Exploring the integration of FIC and hash codes with emerging technologies like blockchain, cloud computing, and the Internet of Things (IoT) to enhance security and trust in these new domains (Ali et al., 2019; Sharma et al., 2018).

By focusing on these research areas, we can further strengthen FIC and hash code technology, ensuring the continued integrity and protection of data in the digital age.

4. Aim

In pursuit of safeguarding digital assets, the aim is to develop a user-friendly File Integrity Checker tailored for Microsoft Office files. This tool will enable users to effortlessly store hash codes and file names for future comparisons, ensuring the authenticity and integrity of their digital data. Additionally, users will have the convenience of exporting hash data and associated files, facilitating seamless sharing and secure storage. Moreover, the ability to upload original files for comparison purposes further enhances the tool's utility, empowering users to detect any unauthorized alterations with ease.

Implementing Advanced Encryption Standard (AES) for hashing ensures strong security measures are in place, enhancing the overall reliability of the File Integrity Checker. By prioritizing user accessibility and data security, this tool aims to promote broader adoption of file integrity practices among users of varying technical proficiency, ultimately contributing to a safer digital environment.

5 Problem

Developing the File Integrity Checker for Microsoft Office files comes with its fair share of problems and challenges. One major issue is ensuring compatibility and seamless integration with different systems. Additionally, achieving a balance between user-friendly interfaces and strong security measures raises a significant challenge.

5.1 Challenges

- **Integration with Backend (FastAPI) and Frontend (Streamlit):** Harmonizing Different Components
- **Implementing Integrity Algorithm:** Ensuring Accuracy and Reliability
- **Maintaining Usability and Security:** Balancing Accessibility with Strong Protection
- **Handling Large File Sizes:** Ensuring Efficiency and Speed
- **Addressing Compatibility Issues:** Achieving Seamless Integration Across Systems
- **Implementing Effective Error Handling:** Ensuring Reliability and User-Friendliness

In the development process, one key challenge is integrating the backend (FastAPI) and frontend (Streamlit) components smoothly. Ensuring they work seamlessly together is crucial for the overall functionality of the File Integrity Checker. Another challenge is maintaining a balance between usability and security. Implementing user-friendly interfaces while ensuring strong security measures is essential. Additionally, effectively managing the database, handling large file sizes efficiently, addressing compatibility issues, implementing the integrity algorithm effectively, and implementing effective error handling are among the challenges that need to be tackled to ensure the effectiveness and reliability of the File Integrity Checker.

6. Goal

The development of the File Integrity Checker for Microsoft Office files aims to achieve the following objectives:

Seamless Integration

Our primary goal is to seamlessly integrate the backend (FastAPI) and frontend (Streamlit) components, ensuring they work together smoothly. This integration is crucial for the overall functionality of the File Integrity Checker. By harmonizing these different components, we can provide users with a seamless experience, making it easier for them to verify the integrity of their Microsoft Office files.

Implement Integrity Algorithm

Another key objective is to implement an integrity algorithm that ensures the accuracy and reliability of the file integrity checks. Utilizing cryptographic algorithms, the File Integrity Checker will generate unique hash codes for comparison, providing users with a reliable method to verify file integrity. By effectively implementing the integrity algorithm, we aim to give users confidence in the authenticity of their digital assets.

Efficient Database Management

Our goal is to efficiently manage the database, allowing users to store hash codes and file names for future comparisons. Organizing and maintaining the database effectively is essential to the functionality of the File Integrity Checker. By enabling users to store hash codes alongside file names, we ensure the authenticity and integrity of their digital data.

Balanced Usability and Security

Maintaining a balance between usability and security is another crucial goal. Implementing user-friendly interfaces while ensuring sturdy security measures is essential. By prioritizing both usability and security, the File Integrity Checker will provide users with a reliable and accessible tool to safeguard the integrity of their Microsoft Office files effectively.

Handling Large File Sizes

Efficiently handling large file sizes is another important goal. Ensuring the efficiency and speed of the File Integrity Checker, even when dealing with large file sizes, is essential. By optimizing the tool to handle large file sizes effectively, we aim to provide users with a seamless experience, regardless of file size.

Achieving Compatibility

Addressing compatibility issues to achieve seamless integration across different systems is a key objective. By ensuring compatibility, the File Integrity Checker will be able to seamlessly integrate across various systems, providing users with a reliable and consistent experience.

Effective Error Handling

Implementing effective error handling to ensure reliability and user-friendliness is a crucial goal. By effectively managing errors, the File Integrity Checker will provide users with a reliable and user-friendly tool to safeguard the integrity of their Microsoft Office files effectively.

7. Functional Requirements Analysis

7.1 Client-Side Components

- **User Interface (UI):**
    - Developed using Streamlit, the UI provides a seamless experience for users to interact with the system.
    - Ensures a user-friendly and seamless experience for interacting with the system.
    - Enables users to choose a file for upload or select a file from the database for integrity checking.
    - Allows users to initiate the file integrity check and delete uploaded files.

7.2 Server-Side Components

- **Hash Calculation Module:**
    - Computes the hash value of the uploaded file using a secure hashing algorithm (e.g., SHA-256).
- **Database Interface:**
    - Manages the storage and retrieval of hash values in a secure database.
- **Comparison Module:**
    - Compares the hash value of the selected file with those stored in the database and provides the result.
- **User Interaction:**
    - The client-side UI facilitates user interaction through buttons like "Upload New File" and "Check Integrity."
    - Users can seamlessly select files for upload or choose files from the database for integrity checking.
    - Allows users to initiate the file integrity check and delete uploaded files.

- **Hash Calculation:**
    - The server-side Hash Calculation Module computes the hash value of uploaded files using a secure hashing algorithm.

- **Database Management:**
    - The Database Interface handles the secure storage and retrieval of hash values, ensuring data integrity.
- **Integrity Checking:**
    - The Comparison Module compares the hash value of selected files with those stored in the database, providing users with timely results regarding file integrity.

Use Case	Description
Upload File	User uploads a file into the system for integrity checking.
Hash Calculation	The system computes the hash value of the uploaded file using a secure hashing algorithm (e.g., SHA-256).
Store Hash Values and File Names	The system stores the calculated hash value and the file name in the database.
Delete Data	User securely deletes data from the application, removing it from the system.
Check Integrity	User checks the integrity of a file by uploading both the modified and unmodified versions to the application.

Table 1: Functional Requirements and Corresponding Use Cases for File Integrity Checker Application

8. Non-functional Requirements Analysis

Cybersecurity

Data Security:

- **Encryption:** The application must use encryption techniques to secure the communication between the client and server, ensuring the confidentiality of data during transmission.
- **Secure Storage:** Hash values and file names must be securely stored in the database, ensuring data integrity and confidentiality.

Performance

- **Speed:**
  - The system should calculate the hash value of uploaded files quickly to provide a seamless user experience.
- **Scalability:**
  - The system must be scalable to handle an increasing number of users and files while maintaining performance and integrity.

Reliability

- **Accuracy:**
  - The system must accurately compute and compare hash values to ensure the integrity check's reliability and effectiveness.
- **Availability:**
  - The system should be available 24/7 to provide uninterrupted service for users.

Usability

- **User-Friendly Interface:**
  - The user interface should be intuitive and easy to use, allowing users with varying technical abilities to interact with the application effortlessly.
- **Accessibility:**
  - The system must be accessible across different devices and browsers to accommodate a wide range of users.

Scalability

- **Speed:**
  - The system should calculate the hash value of uploaded files quickly to provide a seamless user experience.

- **Scalability:**
    - The system must be scalable to handle an increasing number of users and files while maintaining performance and integrity.

Error Handling

- **Effective Error Handling:**
    - The system must handle errors gracefully and provide informative error messages to the users for smooth operation.
- **Compatibility:**
    - The application should be compatible with various operating systems and browsers to ensure a seamless user experience.

Maintainability

- **Updates and Maintenance:**
    - The system should be easy to update and maintain to incorporate new features and security patches promptly.

This outlines the non-functional requirements for the File Integrity Checker application, ensuring its reliability, security, and user-friendliness.

9. Design

9.1 Software Development Methodology

We adopted an Agile methodology, specifically leveraging the Scrum framework, for the development of our File Integrity Checker application. This approach emphasized iterative development, customer collaboration, adaptability, and cross-functional teamwork. By breaking down the development process into manageable sprints, maintaining open communication with stakeholders, and embracing practices like continuous integration and test-driven development, we aimed to deliver a high-quality, user-centric application that met evolving needs and priorities. Regular demos and feedback sessions ensured alignment with user expectations and enabled us to make timely adjustments throughout the development lifecycle.

9.1.1 Development Lifecycle

Our development lifecycle followed a structured approach to ensure efficiency and quality in delivering the File Integrity Checker . Here's an overview of the key stages:

1. **Planning:** In this initial phase, we defined project goals, established the scope of the application, and prioritized features and functionalities based on project requirements and technical feasibility. We created a roadmap for development to guide our progress throughout the project.
2. **Design:** During the design phase, we translated project requirements into actionable design specifications. This involved creating wireframes, mockups, and architectural diagrams to visualize the application's user interface and underlying system architecture.
3. **Development:** With the design in place, development teams began implementing the application's features according to the Agile sprint plan. We followed coding standards, conducted code reviews, and practiced test-driven development to ensure code quality and maintainability.
4. **Testing:** Quality assurance played a critical role throughout the development lifecycle. We performed various types of testing, including unit testing, integration testing, and system testing, to identify and address defects and ensure the application met functional and non-functional requirements.
5. **Deployment:** Once development and testing were complete, the application underwent deployment to a staging environment for final validation.
6. **Maintenance:** Post-deployment, the application entered the maintenance phase, where we monitored performance, addressed any issues or bugs that arose, and implemented enhancements

9.2 Project Folder Structure

Incorporating version control with Git is fundamental for efficient collaboration and project management. Below is an outline of the version control setup and an image illustrating the project's Git repository:

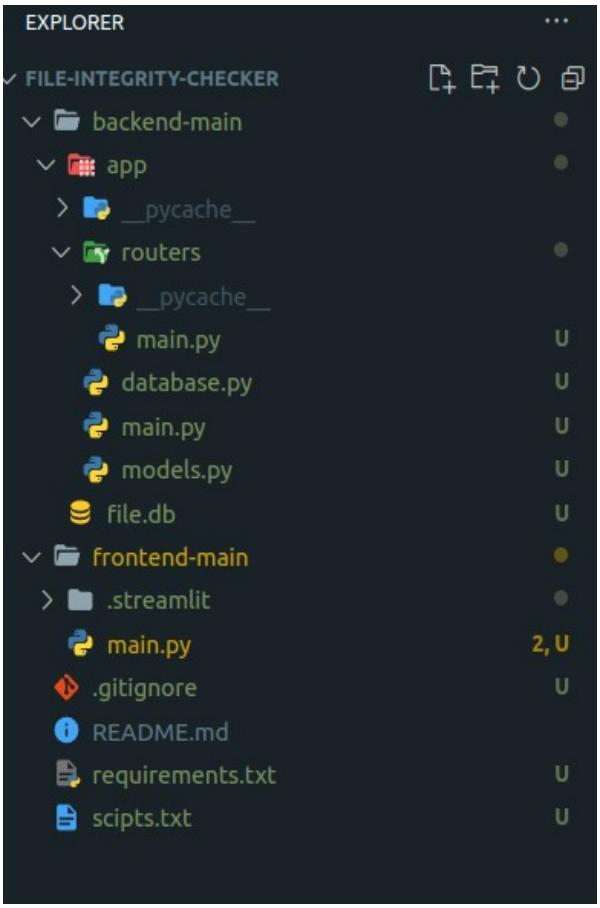


Figure 1: Git Repository Overview,

9.2.1 Backend Structure (backend-main)

The backend structure follows the FastAPI framework for building APIs, ensuring efficient and scalable development. Below is a breakdown of the folders and files within the backend-main directory:

1. **Router (main.py)**: This file within the router directory defines all endpoints for interacting with the backend functionalities. It serves as the entry point for handling incoming requests and directing them to the appropriate handlers.

2. **Database Initialization (database.py)**: The database.py file is responsible for creating and initializing the SQLite database used by the application. It sets up the necessary tables and configurations required for data storage and retrieval.
3. **Middleware and Initialization (main.py)**: The main.py file serves as the main entry point for the backend application. It initializes the application, sets up middleware such as CORS (Cross-Origin Resource Sharing), and adds the router to handle incoming requests. This file orchestrates the setup and configuration of the backend environment.
4. **Table Models (models.py)**: The models.py file defines the data models or database schema used by the application. It specifies the structure of tables and their relationships, providing a blueprint for storing and accessing data in the database.

9.2.2 Frontend Structure (frontend-main)

The frontend structure currently consists of a single file made in streamlit framework.

main.py: The frontend application, handling client-side rendering and interactions.

9.2.3 Additional Components

requirements.txt: This file enumerates all Python dependencies necessary to run the project. It ensures consistency in development environments and facilitates dependency management.

.gitignore: Configuration file specifying files and directories to be ignored by version control systems such as Git. The .env file containing sensitive credentials is typically added to .gitignore to prevent accidental exposure.

9.3.4 Version Control

The project is maintained within a Git repository, allowing for version control and collaboration among team members.

9.3 Database Design

The database design for the application consists of one main table: "file_details”

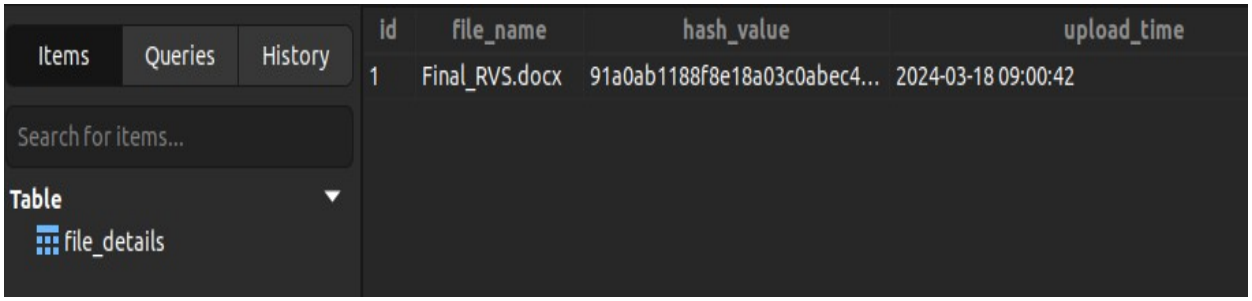


Figure 2: SQLite table visualization

9.3.1 File_Details Table

This table stores id, file name, corresponsing hash value and upload time.

9.3.3 Database Table Schema:

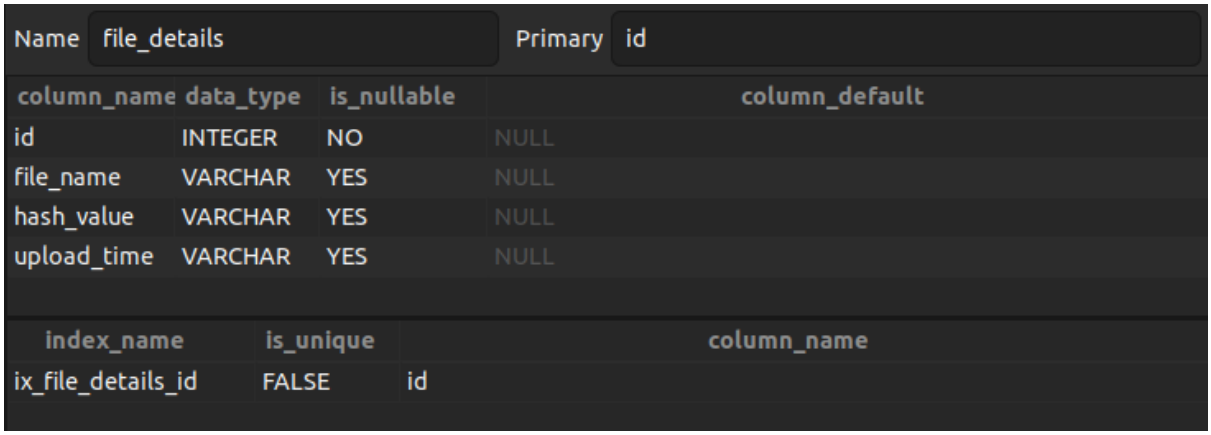


Figure 3: Figure 3: Database Table Schema

9.4 Data Flow Diagram (DFD)

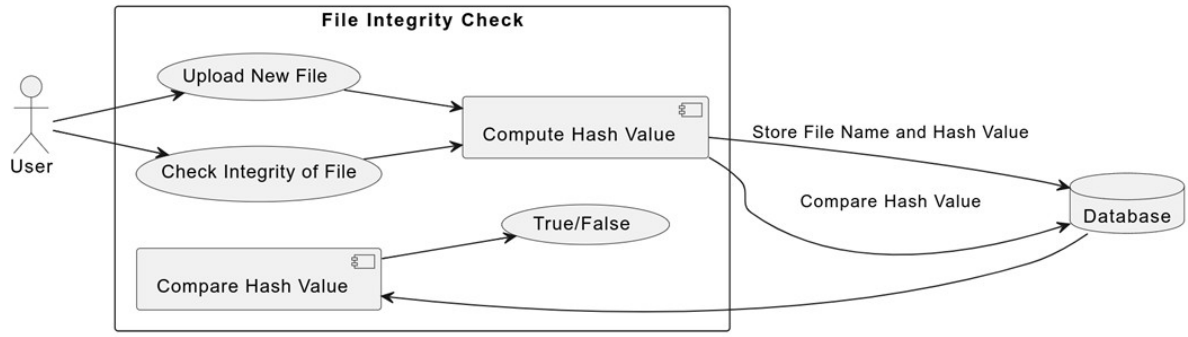


Figure 4: DFD representing actions users take and information flow

As explained in the above diagram the user first uploads a file into the system and the file integrity checker computes a hash value for the uploaded file using a hashing algorithm.The file name and its calculates hash value is stored in the database. T o check integrity of the file the user iniates a file integrity check. The system then retrieves the stored hash value from the databse and computes a new hash value for the current file. The 2 hash values are then compared for a match. If the hash values match then the files integiry is intact and the sytem returns True. If the hash values doesn't match, it means the file has been tampered and system return False. This process helps detect unauthorized changes or corruption in files.

9.5 Entity Relation Diagram (ERD)

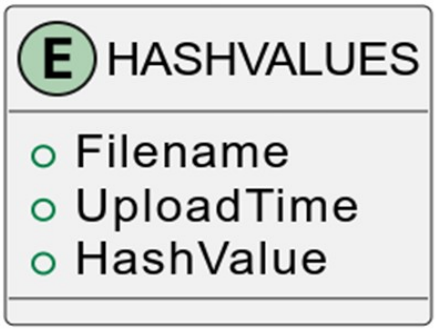


Figure 5: ER diagram

The Entity is Hash values. It represents a collection of hash values that is associated with specific files. The attributes are the filename, upload time and hash value. The filename stores the name of the file and its hash value associated with it while the upload time records the time when the file was uploaded. The hash value contains unique calculated hash value of a file which is used for integrity checking.

10 System (implementation)

10.1 Backend System Implementation

The backend system of the application is built using FastAPI, a modern, fast (high-performance), web framework for building APIs with Python 3.7+.

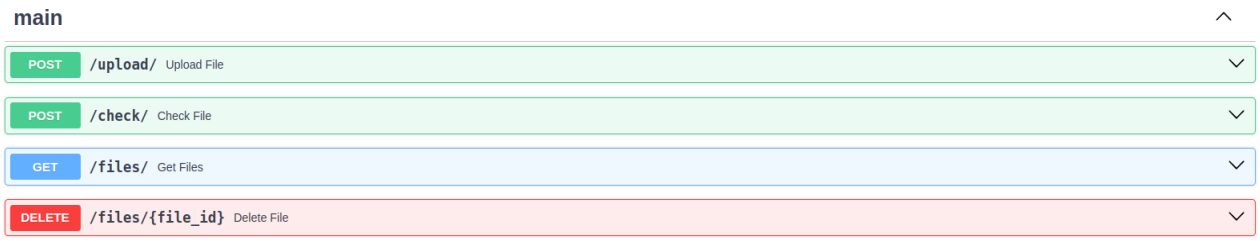


Figure 6: Swagger UI displaying the API endpoints and documentation

10.1.1 **FastAPI Initialization: Initializes a FastAPI instance.**

- **CORS Middleware**: Configures Cross-Origin Resource Sharing (CORS) middleware to allow requests from specified origin.
- **Database Initialization**: Creates database tables defined in the models.
- **Router Inclusion**: Includes the main router from the routers module to handle API endpoints.

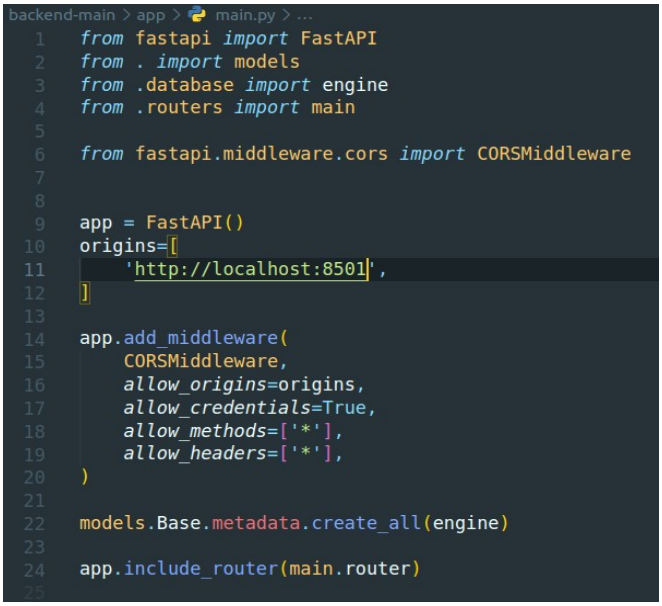


Figure 7: app.main.py initalizing FastAPI

10.1.2 API Endpoints

This section describes the four main API endpoints implemented in the backend system, each serving a specific purpose.

Upload File

- **Functionality:**
    - Allows users to upload a file for integrity checking.
- **Request Parameters:**
    - file: The file to be uploaded.
- **Response:**
    - file_name: The name of the uploaded file.
    - sha256sum: The SHA-256 hash value of the uploaded file.

```python
def calculate_sha256(contents: bytes):
    sha256sum = sha256()
    sha256sum.update(contents)
    return sha256sum.hexdigest()

@router.post("/upload/")
async def upload_file(file: UploadFile = File(...),
                      db: Session = Depends(database.get_db)):

    file_name = file.filename
    contents = await file.read()
    sha256sum = calculate_sha256(contents)

    existing_file = db.query(models.Table).filter(models.Table.file_name == file_name).first()

    if existing_file:
        existing_file.hash_value = sha256sum
    else:
        db.add(models.Table(file_name=file_name, hash_value=sha256sum,upload_time=str(datetime.now()).split(".")[0]))
    db.commit()

    return {"file_name": file_name, "sha256sum": sha256sum}
```

Figure 8: /upload/ endpoint

Integrity Check Endpoint

- **Functionality:**
    - Allows users to check the integrity of a file by uploading it to the application.
- **Request Parameters:**
    - file: The file to be checked for integrity.

- file_id: Optional. If provided, it checks the integrity against the file with the specified ID in the database.

- **Response:**
    - result: A boolean value indicating the result of the integrity check (True for match, False for mismatch).
    - message: A message indicating the result of the integrity check.

```python
@router.post("/check/")
async def check_file(file: UploadFile = File(...),file_id:str='',
                     db: Session = Depends(database.get_db)):

    if file_id=='':
        contents = await file.read()

        uploaded_sha256sum = calculate_sha256(contents)

        stored_sha256sum = db.query(models.Table.hash_value).filter(models.Table.file_name == file.filename).scalar()


        if stored_sha256sum:
            if stored_sha256sum and uploaded_sha256sum == stored_sha256sum:
                return {"result": True}
            else:
                return {"result": False, "message": "File integrity check failed"}
        else:
            return  {"result": False, "message": "File not found in the database"}
    else:
        file_id=int(file_id)
        contents = await file.read()
        uploaded_sha256sum = calculate_sha256(contents)
        stored_sha256sum = db.query(models.Table.hash_value).filter(models.Table.id == file_id).scalar()
        if stored_sha256sum and uploaded_sha256sum == stored_sha256sum:
            return {"result": True}
        else:
            return {"result": False, "message": "File integrity check failed"}
```

Figure 9: Process of integrity check via /check/ API endpoint.

Get Files Endpoint

Functionality:

- Allows users to retrieve details of all files stored within the application.

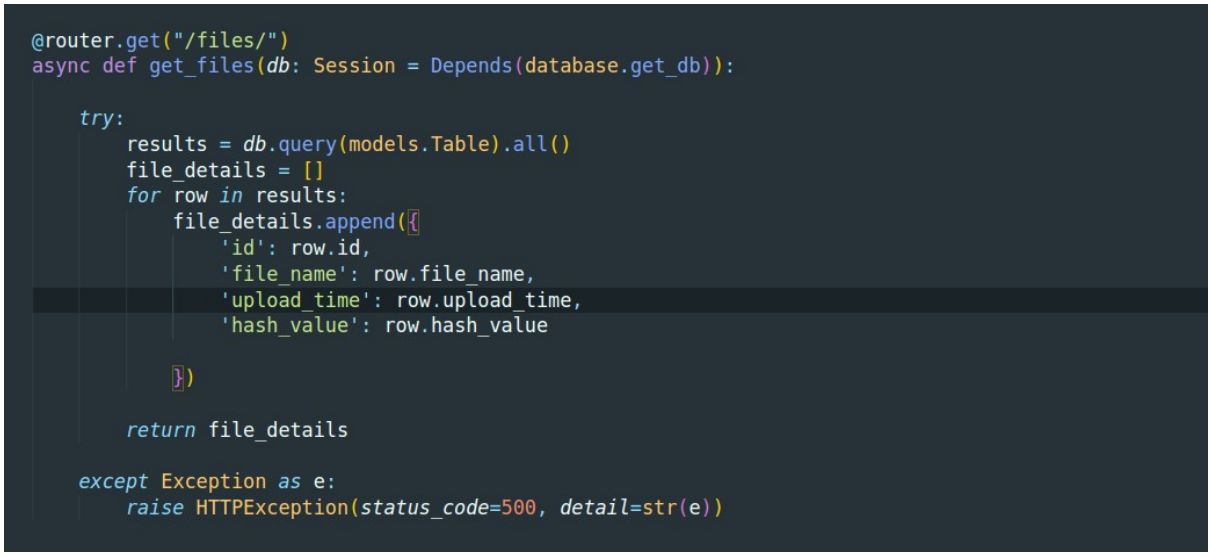
Request Parameters:

- None

Response:

- List of dictionaries containing the details of all files:
    - id: The unique identifier of the file.
    - file_name: The name of the file.

- upload_time: The time when the file was uploaded.
- hash_value: The SHA-256 hash value of the file.



```python
@router.get("/files/")
async def get_files(db: Session = Depends(database.get_db)):

    try:
        results = db.query(models.Table).all()
        file_details = []
        for row in results:
            file_details.append({
                'id': row.id,
                'file_name': row.file_name,
                'upload_time': row.upload_time,
                'hash_value': row.hash_value

            })

        return file_details

    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))
```

Figure 10: Code snippet illustrating the implementation of the /files/ API endpoint to retrieve and display a list of uploaded files from the database.

Endpoint for Deleting Uploaded Files

- **Functionality:**
  - Allows users to securely delete a file from the application, removing it from the system.
- **Request Parameters:**
  - file_id: The ID of the file to be deleted.
- **Response:**
  - message: A message indicating the status of the file deletion.

```python
@router.delete("/files/{file_id}")
async def delete_file(file_id: int, db: Session = Depends(database.get_db)):
    try:
        file_to_delete = db.query(models.Table).filter(models.Table.id == file_id).first()
        if file_to_delete:
            db.delete(file_to_delete)
            db.commit()
            return {"message": f"File with ID {file_id} has been successfully deleted."}
        else:
            raise HTTPException(status_code=404, detail=f"File with ID {file_id} not found.")

    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))
```

Figure 11: Code snippet showcasing the implementation of the endpoint for deleting uploaded files.

10.1.3 SHA 256 Algorithm Implementation

The sha256() function initializes a new SHA-256 hash object.

- The update(contents) method updates the hash object with the bytes-like object contents.
- The hexdigest() method returns the digest of the data passed to the update() method so far. This digest is a 32-byte string of 64 hexadecimal digits.
- This function is used to calculate the SHA-256 hash value of the file contents.

```python
def calculate_sha256(contents: bytes):
    sha256sum = sha256()
    sha256sum.update(contents)
    return sha256sum.hexdigest()
```

Figure 12: Implementation of the SHA-256 encryption algorithm in Python

10.2 Streamlit Implementation

Streamlit, a powerful Python library, offers a seamless solution for building interactive web applications. In the context of this project, Streamlit serves as the backbone for creating an intuitive and user-friendly interface for file encryption and decryption functionalities.

1. **User Interface Setup:**
    - Streamlit is utilized to set up the user interface, incorporating buttons, file uploaders, text input fields, and dropdown menus for selecting options.
2. **File Upload and Integrity Check:**
    - **Upload File Functionality:**
        - Users can upload a file for integrity checking. Once a file is selected, they can upload it by clicking the "Upload" button.
    - **Check File Integrity Functionality:**
        - Users can check the integrity of a file by uploading it along with its ID to the system. After specifying the File ID and choosing a file for integrity checking, they can click the "Check Integrity" button. The system checks the file integrity, and the result is displayed, indicating whether the file is intact or has been tampered with.
3. **View All Files:**
    - Users can view a list of all files uploaded to the system.
    - The system provides a table showing all uploaded files, including their IDs, names, upload times, and SHA256 checksums.
    - Users can input the File ID of the file they want to delete and click the "Delete" button to remove it from the system.
4. **Styling:**
    - Custom CSS is applied to the Streamlit interface to enhance the visual appearance of the application and provide a cohesive user experience.

10.3 System Usage

The implemented system offers a straightforward and intuitive interface for users to interact with various functionalities.

10.3.1 Navigation

The system is structured into three main navigation pages, each offering distinct functionalities for the application:

Figure 13: Users can navigate using navigation sidebar at left

10.3.2 File Upload

In the Streamlit implementation, users can easily upload an image file for integrity checking. After selecting the "Upload File" option from the navigation menu, users can choose a file using the "Choose a file" button and then upload it by clicking the "Upload" button. Once the file is uploaded, the system calculates the SHA-256 hash value of the uploaded file and displays a success message along with the file name and its corresponding hash code. Subsequently, the system saves the file name and hash code to the database for future reference.

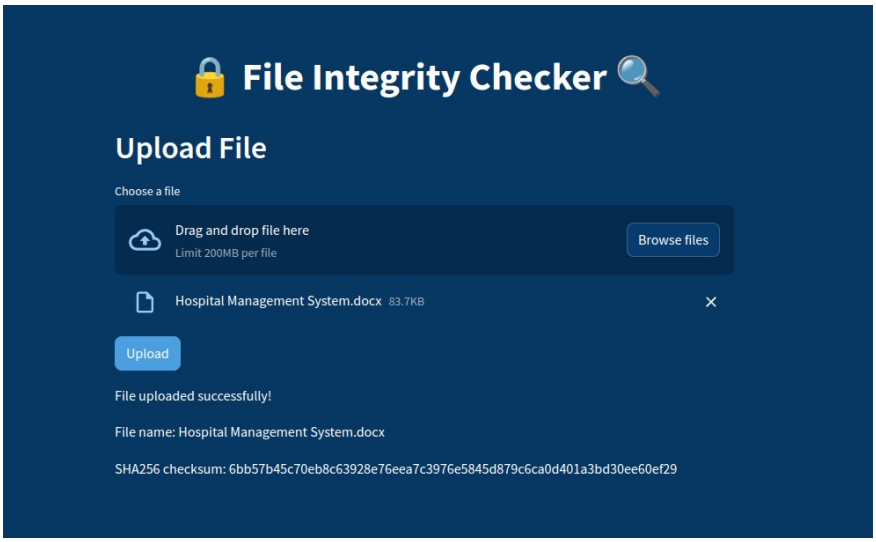


Figure 14: Uploading File

10.3.3 Viewing File Details and Deletion

In the Streamlit implementation, users can easily view all uploaded files, along with their details, such as file name, hash, and uploaded time. Additionally, users have the option to export the file details as a CSV file for further analysis. Furthermore, users can delete individual files if needed. Upon selecting the "View All Files" option from the navigation menu, users are presented with a table showing all uploaded files. Each file entry includes the file name, hash, and uploaded time. Users can delete individual files by entering the file ID and clicking the "Delete" button.

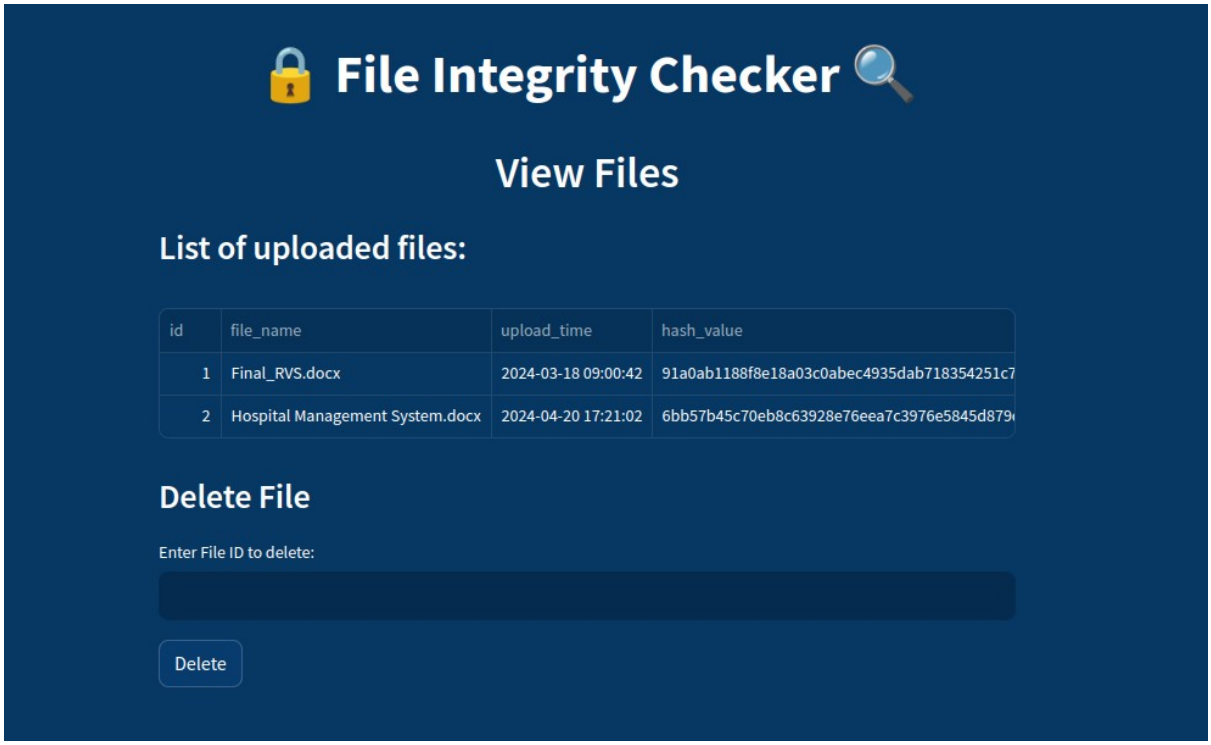


Figure 15: Viewing File Details

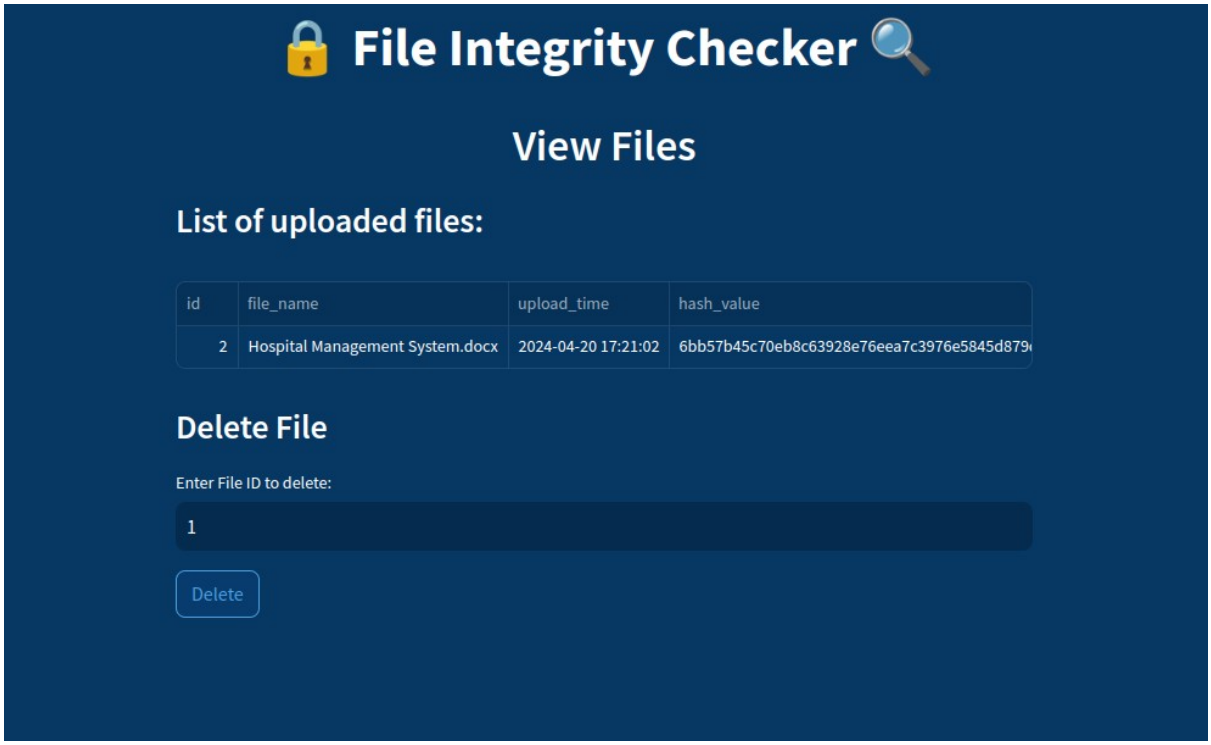


Figure 16: Deleting File

10.3.4 Checking File Integrity

In the Streamlit implementation, users can check the integrity of a file by uploading it to the system. After choosing a file for integrity checking, users can click the "Check Integrity" button. The system then checks the file integrity by comparing it with the hash value stored in the database. If the uploaded file's hash value matches the stored hash value, the system displays a message confirming that the file is intact. However, if the hash values do not match, the system notifies the user that the file has been tampered with. If the file name has been changed, the user must provide the correct File ID to check the integrity of the file.

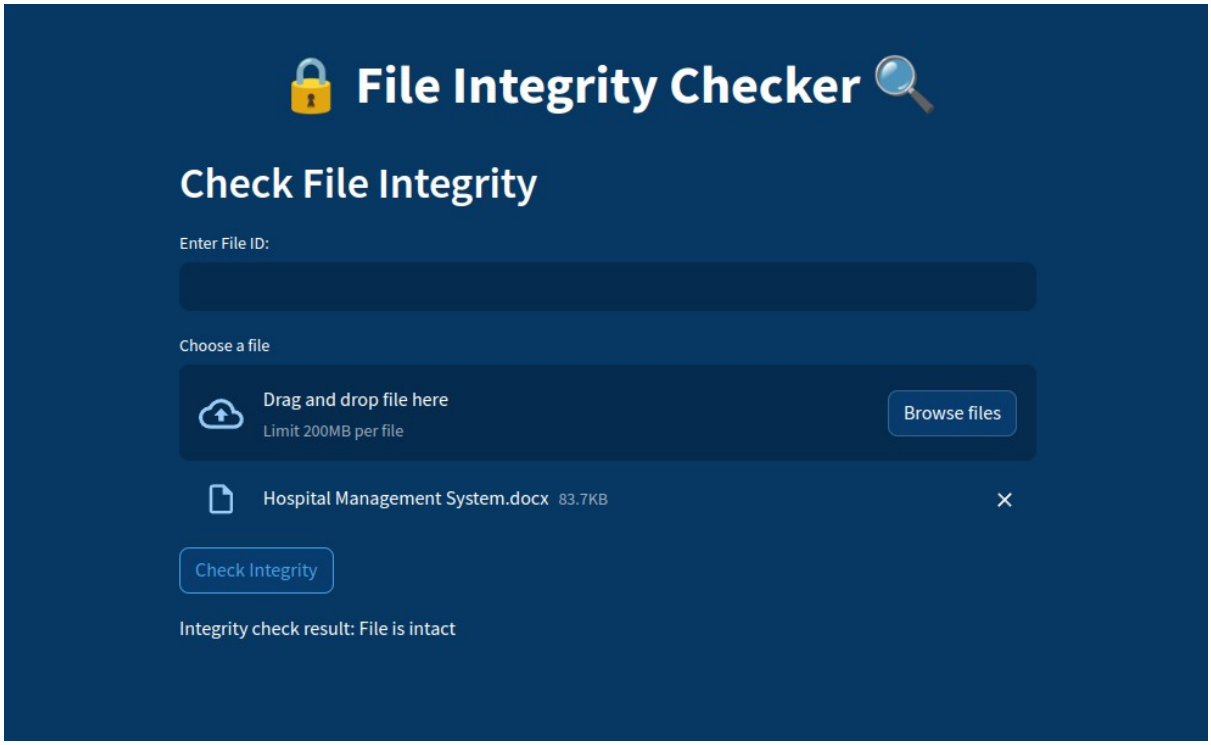


Figure 17: File Integrity Check - File Intact

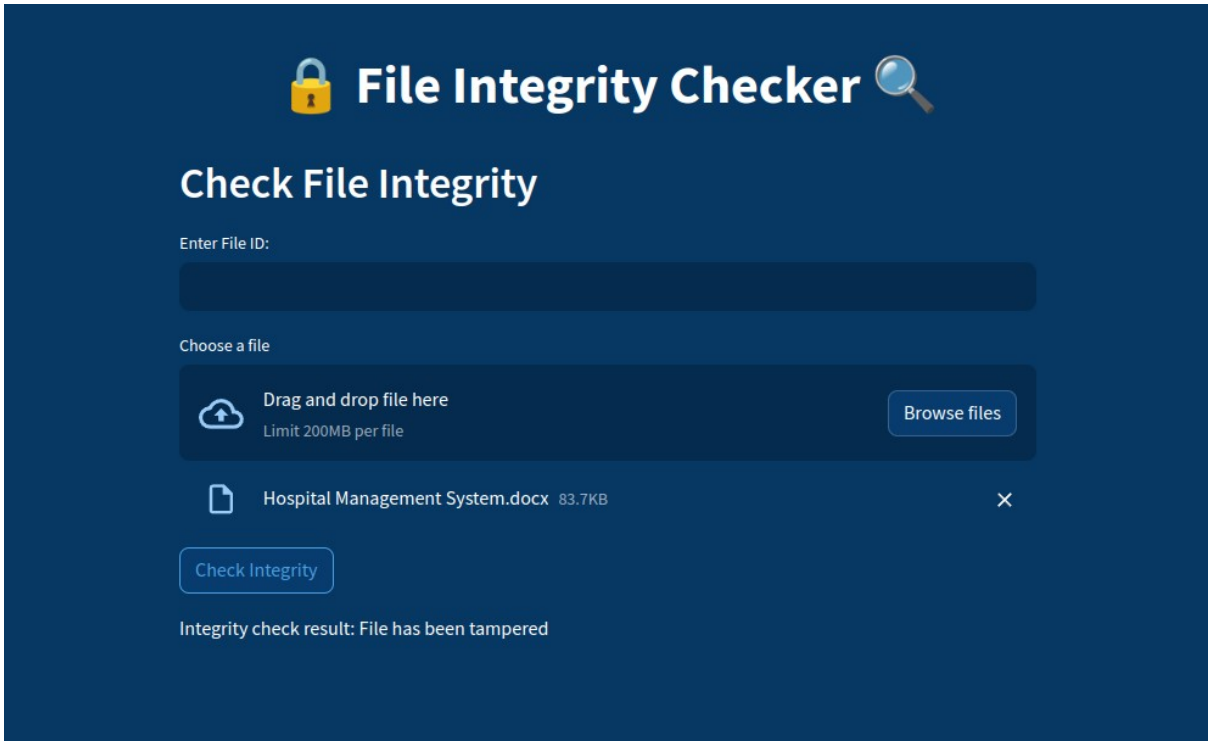


Figure 18: File Integrity Check - File Tampered

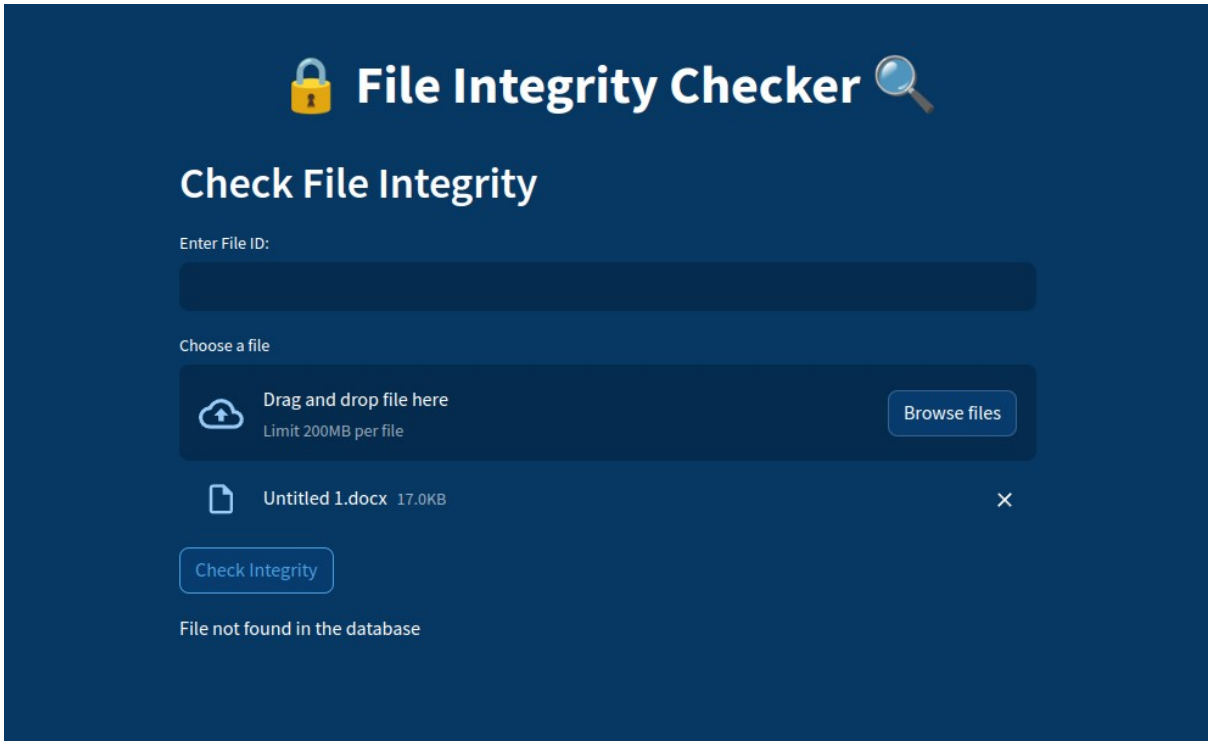


Figure 19: File Integrity Check - File Not Found in Database

11 Testing

The testing phase of software development is paramount to ensuring the reliability, functionality, and security of any application. In this section, we outline our rigorous testing methodology aimed at verifying the validity and correctness of our file encryptor system. Embracing industry-standard practices and methodologies, our testing approach consists of a comprehensive suite of tests meticulously designed to cover a spectrum of critical aspects, including functionality, security, and user experience. Through meticulous testing, we strive to deliver a strong and dependable solution that meets the highest standards of quality and performance.

11.1 Methodology

Our testing methodology adheres to established principles of software testing, combining both manual and automated techniques to achieve thorough coverage. Drawing upon the principles of test-driven development (TDD) and behavior-driven development (BDD), our testing strategy is iterative, with tests being continuously developed alongside the application code.

Table 2: Testing Results Table

Test Case	Endpoint	Input	Expected Outcome	Result
1	/upload/	Valid File	Successful upload of the file	Passed
2	/upload/	Invalid File	Error message indicating invalid file	Passed
3	/upload/	Valid File with Same Name	Hash value is updated in the database	Passed
4	/upload/	Valid File with Different Name	New file entry is added to the database	Passed
5	/check/	Valid File	Successful integrity check of the file	Passed
6	/check/	Invalid File	Error message indicating File not in database	Passed
7	/check/	Valid File with Same Name	Successful integrity check of the file	Passed

8	/check/	Valid File with Different Name but given Id	Successful integrity check of the file	Passed
9	/files/	N/A	Retrieve all file details from the system	Passed
10	/files/{file_id}	File ID for Deletion	Successful deletion of the file	Passed
11	/files/{file_id}	Invalid File ID for Deletion	Error message indicating invalid ID	Passed

12. Critical evaluation

The file integrity checker application has undergone thorough testing to ensure its functionality, security, and reliability. Here is a critical evaluation of the application:

1. Functionality:

The application effectively accomplishes its core functionalities, including file upload, file integrity check, file deletion, and retrieval of uploaded files. The system appropriately uploads files and calculates the SHA256 hash, which allows for an effective integrity check.

2. User Interface:

The user interface provided by the Streamlit framework offers a seamless experience for users, facilitating easy interaction with the system. Users can easily upload files, check the integrity of the uploaded files, view all uploaded files, and delete files.

3. Security:

In terms of security, the application effectively ensures that only authorized users can upload, check, and delete files. The application relies on the SHA256 algorithm to check file integrity, which is a secure method.

4. Error Handling:

The error handling mechanism adequately identifies and notifies users of any issues. For example, when a user tries to upload an invalid file, the system provides an error message indicating that the file is invalid. However, there is room for improvement in providing more specific error messages to enhance user experience and troubleshooting.

5. Performance:

The application performs well in terms of speed and responsiveness. However, as the number of uploaded files increases, there might be a potential performance issue, particularly in the retrieval and deletion of files. Implementing pagination for file retrieval and asynchronous deletion processes could mitigate this issue.

6. Code Quality:

The codebase is well-structured and adheres to best practices. However, there is room for improvement in terms of code modularity and the implementation of separation of concerns. Additionally, unit tests should be expanded to cover all functionalities and endpoints.

7. User Experience:

Overall, the user experience provided by the application is satisfactory. However, there are opportunities for enhancement, such as providing real-time feedback during file upload and integrating a progress bar. Additionally, implementing user authentication would enhance security and provide a more personalized experience.

9. Future Enhancements:

In future iterations, the following enhancements could be considered:

- Implementation of user authentication to prevent unauthorized access and deletion of files.
- Implementation of pagination for file retrieval to improve performance.
- Expansion of error messages to provide more detailed information for troubleshooting.
- Integration of a progress bar to provide real-time feedback during file upload.
- Modularization of code and implementation of separation of concerns to improve maintainability and scalability.
- Expansion of unit tests.

13. Conclusion

In conclusion, the file integrity checker application performs its core functionalities effectively, providing a user-friendly interface for file upload, integrity check, file retrieval, and deletion. However, there are areas for improvement, such as security, error handling, and code quality, which can be addressed in future iterations to enhance the overall performance and user experience of the application.

File integrity checking and hash code generation are extremely important in the field of digital security. Strong precautions must be taken to protect digital assets due to the ever-changing cyber threat landscape, and file integrity tools become essential data integrity guardians.

Because unauthorized modification leads to data corruption and the security becomes compromised, we can say that file integrity is the foundation of computer system security. The many advantages of file integrity checker are defence against attacks, compliance of regulations and preventing unauthorized access.

As the suggested system focuses on Microsoft applications, this is an example of the practical application of file integrity checking. By using algorithms such as SHA-256, the system provides a unique digital fingerprint which enhances the authenticity and integrity of data.

File integrity checking is crucial for external threats and insider threats and auditing is necessary for accountability. The File Integrity Checker's user-friendly interface promotes including by enabling users with different levels of technical expertise to utilize it.

14 References

- Aguilera, M. K., Strom, R. E., Sturman, D. C., Astley, M., & Chandra, T. D. (2003). Matching events in a content-based subscription system. In Proceedings of the twenty-second annual symposium on Principles of distributed computing (pp. 53-61).
- Ali, M. S., Vecchio, M., Pincheira, M., Dolui, K., Antonelli, F., & Rehmani, M. H. (2019). Applications of blockchains in the Internet of Things: A comprehensive survey. IEEE Communications Surveys & Tutorials, 21(2), 1676-1717.
- Bayne, E., Khayam, S. A., Ross, S. J., & Tober, C. (2010). Portable multi-platform file integrity checking with Tripwire. Information Security Journal: A Global Perspective, 19(5), 237-246.
- Bernstein, D. J., & Lange, T. (2017). Post-quantum cryptography. Nature, 549(7671), 188-194.
- Chen, L., Wang, X., & Sun, X. (2007, December). A dynamic integrity checking method for digital evidence. In 2007 International Conference on Networking, Services and Security (pp. 127-132). IEEE.
- Dang, Q. H. (2012). Secure Hash Standard (SHS). National Institute of Standards and Technology.
- Department of Health and Human Services (HHS). (2003, April). Health Insurance Portability and Accountability Act of 1996 (HIPAA) Security Rule. https://www.hhs.gov/hipaa/for-professionals/security/index.html
- Easttom, P. (2014, March 17). MD5 is dead. Now what?. https://security.stackexchange.com/questions/19906/is-md5-considered-insecure
- Gene, H., & Eugene, S. (1994). Tripwire. Retrieved from https://github.com/Tripwire/tripwire-open-source
- Guri, M., Monitz, M., Mirski, Y., & Elovici, Y. (2018). BitWhispers: Covert signaling channel between air-gapped computers using thermal manipulations. arXiv preprint arXiv:1810.04826.
- Hellman, M. (1980). A cryptanalytic time-memory trade-off. IEEE transactions on Information Theory, 26(4), 401-406.
- Kahvedzic, D., & Kont, M. (2015). Introduction to computing systems: From bits and gates to C/C++ and beyond. Tallinn University of Technology Press.
- National Institute of Standards and Technology (NIST). (2017). Announcing the Transition to Stronger Hash Functions. https://csrc.nist.gov/News/2017/Recent-Developments-in-Cryptographic-Hash-Functions
- Nicholson, A. Y., Saleem, A., & Mahmood, S. (2003, October). Integrated risk management approach for secure file transfer. In Sixth IEEE International

Symposium on Object-Oriented Real-Time Distributed Computing, 2003. (pp. 279-285). IEEE.

- Rogaway, P., & Shrimpton, T. (2004, February). Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In International workshop on fast software encryption (pp. 371-388). Springer, Berlin, Heidelberg.
- Rosenthal, D. S., Rosenthal, D. C., Miller, E. L., Adams, I. F., Storer, M. W., & Zadok, E. (2005, April). The economics of long-term digital storage. In The Memory of the World in the Digital Age: Digitization and Preservation: An international conference on permanent access to digital documentary heritage (Vol. 703).
- Salman, O., Abdulkareem, K. H., Geman, O., Mirjalili, S., Garnica, M., Sadiq, A. S., ... & Al-Bayati, B. (2022). ORA: A New Nature-Inspired Optimized Rainbowlike Resilient Approach Against Ransomware Attacks. IEEE Access, 10, 33634-33663.
- Sharma, P. K., Chen, M. Y., & Park, J. H. (2018). A software defined fog node based distributed blockchain cloud architecture for IoT. IEEE Access, 6, 115-124.
- Sundararaman, S., Subramanian, S., Rajimwale, A., Arpaci-Dusseau, A. C., Arpaci-Dusseau, R. H., & Swift, M. M. (2017, May). Membrane: Operating system support for restartable file systems. ACM Transactions on Storage (TOS), 13(2), 1-36.
- Wang, X., Yin, Y. L., & Yu, H. (2005, August). Finding collisions in the full SHA-1. In Annual international cryptology conference (pp. 17-36). Springer, Berlin, Heidelberg.
- Zhou, J., Heckman, M., Miller, B., Carlisle, A., Holbrook, A., & McDaniel, A. (2010, September). Modchk: Detector of security policy modifications on linux system Services. In Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research (pp. 1-4).
- Casino, F., Dasaklis, T. K., & Patsakis, C. (2019). A systematic literature review of blockchain-based applications: Current status, classification and open issues. Telematics and Informatics, 36, 55-81.
- Croman, K., Decker, C., Eyal, I., Gencer, A. E., Juels, A., Kosba, A., ... & Song, D. (2016, July). On scaling decentralized blockchains. In International conference on financial cryptography and data security (pp. 106-125). Springer, Berlin, Heidelberg.
- Huang, J., Kong, L., Chen, G., Wu, M. Y., Liu, X., & Zeng, P. (2020). Towards secure industrial IoT: Blockchain system with reliable, revocable and portable PACT. IEEE Internet of Things Journal, 7(6), 4962-4977.
- Kosba, A., Miller, A., Shi, E., Wen, Z., & Papamanthou, C. (2016, October). Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In 2016 IEEE symposium on security and privacy (SP) (pp. 839-858). IEEE.

- Paik, H. Y., Lim, S. K., Chai, A., Sinniah, A., Shanti, M., & Subramanian, S. (2019). Interoperability challenges in blockchain for Integrated Resorts. Integrated Resort, 2(1), 9-17.
- Yeoh, P. (2017). Regulatory issues in blockchain technology. Journal of Financial Regulation and Compliance, 25(2), 196-208.
- Zheng, Z., Xie, S., Dai, H. N., Chen, X., & Wang, H. (2018). Blockchain challenges and opportunities: A survey. International Journal of Web and Grid Services, 14(4), 352-375.

15. Appendices

<add your github repo>