International Journal of Science and Research Archive

eISSN: 2582-8185

Cross Ref DOI: 10.30574/ijsra

Journal homepage: https://ijsra.net/

(REVIEW ARTICLE)

Check for updates

Implementation of file hashing and verification using HMAC-SHA3

Sikirat Damilola Mustapha * and Bharath Kumar Samanthula

School of Computing, College of Science and Mathematics, Montclair State University, Montclair, New Jersey, United States of America.

International Journal of Science and Research Archive, 2024, 13(02), 3467-3476

Publication history: Received on 06 November 2024; revised on 14 December 2024; accepted on 16 December 2024

Article DOI: https://doi.org/10.30574/ijsra.2024.13.2.2490

Abstract

Maintaining file integrity is crucial for security and operational dependability in industries like transportation infrastructure that depend on accurate data. This work introduces a File Integrity Management System (FIMS) that provides tamper-evident digital file verification using Hash-based Message Authentication Codes (HMAC) with SHA-3. This system is implemented in Java and consists of two main programs: VerifyHash compares computed and stored hashes to detect unauthorized file changes, and CreateHash produces and stores unique HMAC-SHA3 hashes for files. In high-security settings where undetected file modifications could jeopardize safety, compliance, and trust, FIMS provides a dependable method to confirm file integrity. The system's strong architecture and focus on data security make it highly applicable to sectors that value data authenticity, highlighting the significance of safe file management procedures.

Keywords: File Integrity Management; HMAC; SHA-3; Data Integrity Verification; Cryptographic Hashing; Secure File Verification

1. Introduction

In many industries, particularly those involving vital infrastructure like banking, healthcare, and transportation, the integrity of digital files is fundamental to security, dependability, and confidence. It is crucial to make sure that files don't change from their original, verified condition in settings where data manipulation or illegal changes could have disastrous results. For instance, corrupted data may impede safety procedures, operational controls, and crucial decision-making processes in transportation infrastructure, all of which may influence service dependability and public safety(NIST, 2014).

Traditional techniques for verifying file authenticity are no longer adequate to satisfy the security requirements of these high-stakes situations as cyber-attacks continue to get more complex. To proactively detect and prevent tampering or illegal modifications, it is now essential to develop strong file integrity management techniques (Oracle, 2018). This project presents a File Integrity Management System (FIMS) that uses the SHA-3 hashing method in conjunction with Hash-based Message Authentication Codes (HMAC). This method, which was created to offer a trustworthy verification mechanism, lets users create, save, and validate distinct cryptographic hashes for files, making it easier to quickly identify material that has been changed or corrupted.

This project responds to the increasing demand for data security and authenticity in industries where file integrity breaches could have serious repercussions by providing a safe and effective way to manage file integrity. A robust layer of cryptographic security is added by using HMAC-SHA3, guaranteeing that the system can withstand manipulation and preserve data dependability. Organizations can promote operational stability, regulatory compliance, and public trust by bolstering their cybersecurity posture with this system (NIST, 2014).

* Corresponding author: Sikirat Damilola Mustapha

1.1. Problem statement

Data integrity is a crucial aspect of cybersecurity in today's digital environment, particularly in industries where data corruption or unauthorized change can have dire repercussions. The dangers of compromised file integrity have increased due to the growing complexity and interconnectedness of systems in the healthcare, financial, and transportation infrastructure sectors. Even little illegal file changes within these industries can result in serious interruptions, security flaws, and possible safety risks (Oracle, 2018). Keeping files safe from manipulation is crucial to preserving both public confidence and operational stability.

For high-stakes applications, conventional methods of file verification—like simple checksums or digital signatures—are frequently inadequate. These techniques might not be reliable enough to identify subtle or complex manipulation efforts, leaving vital systems open to unnoticed modifications. Organizations also struggle to achieve regulatory compliance standards, which frequently call for stringent data integrity processes to safeguard sensitive information, in the absence of a safe and trustworthy verification mechanism (NIST, 2014).

To meet these urgent needs, this project uses the SHA-3 hashing technique to create a File Integrity Management System (FIMS) that makes use of Hash-based Message Authentication Codes (HMAC). The HMAC-SHA3 technique offers a cryptographically safe way to confirm the validity of files, providing greater resistance to alteration than traditional verification methods. The technology aids in the effective and efficient detection of any unauthorized changes by creating distinct hash values for every file and enabling comparison at a later time. In industries like transportation, where preserving data integrity is essential for compliance, safety, and dependability, this capacity is especially pertinent.

In conclusion, by putting in place a system that creates, saves, and validates distinct HMAC hashes for files, this project responds to the increasing demand for a reliable file integrity management solution. Given the growing risk of cyberattacks and the possible consequences of stolen data, a safe and effective file integrity system is necessary to enable safe operations in high-stakes situations. The proposed FIMS intends to improve data security, regulatory compliance, and operational stability across numerous crucial sectors by offering a dependable way to identify unauthorized modifications.

Objective

The primary objective of this project is to develop a secure and reliable File Integrity Management System (FIMS) that provides effective protection against unauthorized modifications to critical files. This system is designed to detect and report any tampering by generating and verifying cryptographic hashes using the HMAC-SHA3-256 algorithm. The project aims to accomplish this through two core programs:

- **CreateHash Program**: The first program is responsible for generating a unique HMAC-SHA3-256 hash for each file in a specified source directory. This hash, functioning as a cryptographic fingerprint, is stored securely in a designated destination directory. By creating and maintaining these unique hash records, the system establishes a baseline for the original state of each file, enabling future integrity checks.
- **VerifyHash Program**: The second program verifies file integrity by recalculating the HMAC-SHA3-256 hash for each file in the source directory and comparing it to the stored hash value. If the hashes match, it confirms that the file remains unaltered. If there is a discrepancy between the stored and recalculated hashes, the system flags the file as modified, alerting users to potential unauthorized changes.

The overarching goal of the project is to provide organizations with a dependable method for detecting unauthorized file changes, thereby supporting secure data management practices and enhancing data integrity in high-stakes environments. By implementing a cryptographically secure file integrity solution, this project seeks to assist organizations in mitigating risks associated with data tampering, supporting regulatory compliance, and maintaining trust in the authenticity of their data.

1.2. Implementation Details Programming Language and Environment

The File Integrity Management System was developed in Java, utilizing the IntelliJ IDEA development environment. Java was chosen for its portability, robust standard library, and strong cryptography support, which make it suitable for applications requiring security and cross-platform functionality (NIST, 2014). Java's ability to perform well across different platforms ensures the File Integrity Management System is adaptable to various environments, from individual systems to enterprise-level deployments.

1.2.1. Libraries and Modules

Cryptography Libraries

The cryptographic implementation utilizes:

- java.security.NoSuchAlgorithmException for handling the SHA-3 hashing algorithm.
- javax.crypto.Mac and javax.crypto.spec.SecretKeySpec for HMAC generation, which adds an additional layer of authentication to the hashing process.

File Handling Libraries

File operations were managed using:

- java.io.File and java.nio.file.Files libraries, which facilitated efficient and reliable handling of file input/output operations, a critical component when dealing with multiple or large files.

Program 1: Hash Generation (CreateHash Class)

The CreateHash program's primary function is to generate HMAC-SHA3-256 hashes for files. This program takes command-line inputs for source and destination directories, computes the HMAC-SHA3-256 hash for each file in the specified directory, and stores the generated hash in a corresponding file within the destination directory.

Key Steps in CreateHash Class

- **Command-Line Argument Parsing**: Command-line arguments specify source and destination directories for hash generation and storage.
- **HMAC-SHA3-256 Hash Calculation**: The HMAC is computed for each file's contents to ensure data integrity.
- **Hash Storage**: Each computed hash is stored in the destination directory, creating a unique hash record for each file.

Pseudo-code snippet:
java

```
File sourceDir = new File(args[0]);

File destinationDir = new File(args[1]);

for (File file : sourceDir.listFiles()) {

 byte[] content = Files.readAllBytes(file.toPath());

 byte[] hash = generateHMACSHA3(content);

 Files.write(new File(destinationDir, file.getName() + ".hash").toPath(), hash);

}
```

Program 2: Verification (VerifyHash Class)

The VerifyHash program recalculates the HMAC-SHA3-256 hash for each file and compares it to the stored hash, enabling users to validate file integrity. Command-line inputs specify the source directory (containing files) and destination directory (containing stored hashes).

Key Steps in VerifyHash Class:

- **Hash Comparison**: For each file, the stored hash is compared with a newly computed hash.
- **Output**: The program outputs each filename followed by "YES" if the hashes match or "NO" if they do not, allowing users to detect any alterations.

Pseudo-code snippet
java

```
for (File file : sourceDir.listFiles()) {

 byte[] content = Files.readAllBytes(file.toPath());

 byte[] storedHash = Files.readAllBytes(new File(destinationDir, file.getName() + ".hash").toPath());

 byte[] currentHash = generateHMACSHA3(content);

 System.out.println(file.getName() + (Arrays.equals(storedHash, currentHash) ? " YES" : " NO"));

}
```

1.3. Importance and Functionalities of the File Integrity Management System

Unauthorized file changes can have serious repercussions, including operational disruptions, financial losses, and compromised safety, in settings where data integrity is crucial, such as financial institutions, healthcare systems, and transportation infrastructure (NIST, 2014). Maintaining the reliability of data requires file integrity verification, which makes sure that files don't change from their initial, validated condition. These issues are addressed by the File Integrity Management System (FIMS), which was used in this project and offers an organized way to keep an eye on and verify the legitimacy of files.

Because illegal file changes can affect public safety and interfere with vital operations, file integrity control is especially important in the infrastructure industry. In order to meet these requirements, this project implements a verification system that uses SHA-3 and Hash-based Message Authentication Codes (HMAC) to offer a tamper-evident, cryptographically secure layer for data protection. HMAC-SHA3 provides efficiency and security, guaranteeing that the system is dependable in shielding private data from unauthorized changes.

The File Integrity Management System's features There are two main features built into the FIMS design:

- **Hash Generation:** FIMS creates a distinct HMAC-SHA3-256 hash for every file in a designated source directory using the CreateHash application. This hash serves as the digital fingerprint of the file, enabling dependable storage for comparison at a later time. The generated hashes are safely kept in a specified directory, guaranteeing their integrity while making them available for verification.
- 2.**Hash Verification:** Each file in the source directory gets its HMAC hash recalculated by the VerifyHash program, which then compares it to the stored hash. It verifies that the file hasn't been changed after the hash was created if the hashes match. The system notifies users of possible unauthorized changes by marking the file as changed if the hashes do not match.

These features offer a thorough method of managing file integrity, empowering users to recognize and react to file modifications efficiently.

1.3.1. FIMS Applications and Suggested Environments

The FIMS can be suggested for a number of applications because to its strong structure, especially in fields where data integrity is crucial:

- **Transportation Infrastructure Security:** To avoid possible safety hazards or operational interruptions, data integrity must be maintained as transportation systems depend more and more on digital controls and information systems. In order to ensure that any unauthorized changes are quickly identified, FIMS can be used to monitor configuration files, security logs, and other important data within transportation networks.

  Healthcare Systems: Accurate and secure patient data and treatment information are essential in healthcare settings. By putting FIMS into place, clinics and hospitals can ensure that electronic health records (EHRs) are accurate and reliable for clinical decision-making.
- **Financial Services:** By using FIMS to keep an eye on transaction logs, audit files, and regulatory data, financial institutions may guard against unauthorized changes that might result in compliance violations or financial inaccuracies.

- **Corporate IT Environments:** By using FIMS to preserve the integrity of data backups, configuration files, and important documents, businesses that handle confidential client information and proprietary data may guarantee consistency and dependability in their systems.

The File Integrity Management System's Effect In high-stakes situations, the FIMS can greatly increase data security and trust. This technology enables companies to quickly identify unauthorized file changes by offering a way to validate file integrity. Implementing FIMS has several important effects, including:

- **Improved Data Security:** By adding a tamper-evident layer to data management, FIMS assists enterprises in shielding confidential information from unwanted modifications. Critical data can be kept safe and the chance of data breaches can be decreased with this extra layer of protection.
- **Operational Reliability:** By guaranteeing that crucial files stay authentic, FIMS contributes to system reliability in industries like healthcare and transportation where data integrity has a direct influence on operations. The continuous provision of services depends on this dependability.
- **Regulatory Compliance:** Strict data security regulations must be followed by a number of businesses, including healthcare and banking (NIST, 2014). By giving businesses, a methodical approach to confirming file integrity, FIMS promotes compliance and helps them fulfill legal obligations while avoiding possible fines for non-compliance.
- **Enhanced Trustworthiness**: By putting in place a file integrity management system, a company may reaffirm its dedication to data security and gain the confidence of stakeholders, clients, and authorities. Businesses are more likely to win over partners and clients if they can show that their data integrity procedures are working.

2. Security consideration

Since the File Integrity Management System (FIMS) is intended to identify unwanted file alterations in settings where data integrity is crucial, its security is of the utmost importance. Several security measures and principles were integrated into the design and implementation of the system to guarantee its resilience:

2.1. Choice of Cryptographic Hashing Algorithm

The FIMS leverages the SHA-3-256 hashing algorithm within a Hash-based Message Authentication Code (HMAC) framework, known for its resistance to collision and preimage attacks (NIST, 2014). SHA-3 is particularly suitable for applications requiring high security, as it is less susceptible to vulnerabilities found in older hashing algorithms like SHA-1 or MD5, which are prone to collision attacks. By integrating SHA-3, the system benefits from enhanced resistance to cryptographic attacks that could compromise the reliability of hash values.

2.2. HMAC Implementation for Improved Authentication

By requiring a secret key during hash generation, the usage of HMAC (Hash-based Message Authentication Code) provides an extra layer of authentication to the hashing process. Even if an attacker manages to obtain the hash values, this key-based technique makes sure that they are difficult to recreate or change without the secret key. Because only authorized users with the secret key may produce legitimate hashes, this feature adds another degree of protection against tampering (Oracle, 2018).

2.3. Important Management Techniques

For HMAC-based systems to remain secure, proper key management is essential. Secret keys are safely stored in the FIMS and are neither directly incorporated in the source code nor hard-coded. To lower the risk of exposure, they can be safely kept in an environment variable or separate configuration file with limited access. This method guarantees that hash creation and verification stay safe while also preserving the integrity of the HMAC process.

2.4. Data Integrity and Confidentiality

The system places a high priority on cryptographic operations as well as file data security and integrity. Preventing illegal changes indirectly supports secrecy, even if integrity verification is the main objective. Additionally, the system uses Java's java.nio.file.Files library to handle files securely, guaranteeing dependable and strong file input/output operations. The system is set up to verify each file's integrity without needlessly disclosing private information or hash values.

2.5. Exception Management and Error Handling

In order to handle any operational problems like erroneous file paths, missing directories, or missing hash files, strong error handling and exception management were put in place. The system is protected against interruptions that may otherwise expose it to possible security threats by managing exceptions such as NoSuchAlgorithmException, IOException, and InvalidKeyException. The system makes sure that error reporting doesn't create risks by logging important information in the event of an error without jeopardizing sensitive data.

2.6. Defense against Attacks by Replay

The FIMS makes sure that every hash value is directly linked to the distinct content of every file and the particular HMAC key in order to reduce the possibility of replay attacks, in which previously recorded data could be reused to jeopardize system integrity. This method ensures that every verification is distinct and only valid for the file and configuration by making it impossible to replay or reuse an old hash without the precise original file content and key.

2.7. Security Assurance Testing

To ensure the security of the hashing and verification procedures, extensive testing was carried out. To replicate real-world situations and assess the system's resistance to illegal file changes, both unit and integration tests were created. Through these testing, the system's ability to accurately identify altered files under a variety of operating settings was confirmed.

2.8. Flexibility and Scalability at Different Security Levels

The FIMS is appropriate for a range of security needs because it was created with scalability and adaptability in mind. Access restrictions can be put in place to restrict access to important components, and the HMAC key can be changed on a regular basis for businesses that need higher security. These extra security features facilitate the system's use in settings including financial organizations, healthcare facilities, and transportation that have strict security regulations.

The FIMS offers a safe and dependable file integrity management solution by taking these security factors into account. These safeguards guarantee that the system can withstand unwanted file modifications, preserve the integrity of important files, and be modified to satisfy strict security standards in sensitive industries like critical infrastructure.

3. Testing and validation

To ensure the functionality, security, and reliability of the File Integrity Management System (FIMS), a structured approach was taken for testing and validation. Testing focused on unit tests for core components, integration tests for system functionality, and security validation to ensure resilience against tampering and unauthorized modifications.

3.1. Unit Testing

Unit tests were performed on the main functions of both the **CreateHash** and **VerifyHash** classes. These tests aimed to verify that each function performs as expected in isolation, ensuring accurate and consistent hash generation and verification.

- **Hash Generation**: The CreateHash class was tested to confirm that each file processed generates a unique and consistent HMAC-SHA3-256 hash. Multiple test files with different content types and sizes were used to verify the uniqueness and accuracy of generated hashes.
- **Hash Storage**: After generating hashes, the system was tested to ensure they were correctly stored in the designated directory without errors, confirming that the hash files were accessible for verification.
- **Hash Verification**: The VerifyHash class was tested to compare recalculated hashes with stored values. The system was expected to return "YES" for unmodified files and "NO" for any modified files, accurately reflecting changes.

Results: Unit testing confirmed that the core functions—hash generation, storage, and verification—operate accurately and consistently across a variety of files. No issues were observed in the generation or comparison of hash values.

3.2. Integration Testing

Integration testing was conducted to validate the end-to-end functionality of the FIMS, ensuring that all components work together as intended. Different scenarios were created to test the system's ability to detect file integrity violations under realistic conditions.

- **Scenario 1: Unmodified Files** A set of unaltered files was processed through both the CreateHash and VerifyHash programs. The goal was to confirm that the system outputs "YES" for all files, indicating no unauthorized changes. *Result*: The system successfully identified all unmodified files as intact, validating its ability to confirm file integrity when no alterations are present.
- **Scenario 2: Modified Files** A set of files was modified after the initial hash generation. The VerifyHash program was then run to compare hashes. This test aimed to verify that the system detects any file discrepancies. *Result*: The system accurately flagged all modified files, outputting "NO" for each altered file, thus validating its effectiveness in detecting unauthorized file modifications.
- **Scenario 3: Large Files and Batch Processing** To test performance under high-volume conditions, the system processed large files (multi-megabyte files) and ran batch operations on directories with numerous files. *Result*: The FIMS maintained consistent performance across large files and batch operations, demonstrating its scalability for high-data environments without sacrificing accuracy or efficiency.
- **Scenario 4: Error Handling for Invalid Inputs** Testing included scenarios with invalid command-line arguments, nonexistent directories, and missing files to evaluate the system's error management. *Result*: The system handled all errors gracefully, providing informative error messages without crashing, ensuring that incorrect inputs do not disrupt system stability or security.

3.3. Security Validation

Security validation focused on ensuring the resilience of the FIMS against tampering attempts, unauthorized key access, and maintaining the confidentiality of hash values and HMAC keys.

- Key Confidentiality: Tests were conducted to confirm that the HMAC key was stored securely, avoiding exposure in code, logs, or outputs, and could not be accessed by unauthorized users.
- Tamper Resistance: Attempts were made to modify stored hash files and to substitute the original hash files with altered versions. The system's ability to detect these modifications through the VerifyHash function was a critical aspect of the security validation process.

Result: Security validation confirmed that the system maintains key confidentiality and reliably detects tampering, ensuring that file integrity checks remain secure even in the presence of potential threats.

3.3.1. Summary of Testing and Validation Results

The testing and validation process demonstrated that the FIMS performs accurately, securely, and efficiently, confirming its suitability for high-security environments.

- Accuracy: The system consistently generated and verified hashes, reliably identifying unmodified and modified files without any false positives or negatives.
- Efficiency: Performance was maintained during batch processing and with large files, proving that the FIMS is efficient enough for data-intensive applications.
- Robust Error Handling: The FIMS handled incorrect inputs and error-prone conditions effectively, ensuring user feedback without exposing sensitive data or compromising system stability.
- Security Resilience: The FIMS upheld key confidentiality and showed tamper resistance, confirming that integrity checks remain reliable under various conditions.

This rigorous testing and validation process affirms that the FIMS provides an effective and secure solution for file integrity management in critical sectors, such as transportation infrastructure, healthcare, and finance, where data reliability and security are paramount.

4. Challenges

The implementation of strong cryptography and error-handling methods, as well as the effective handling of big files and batch processes, were two major problems in the creation of the File Integrity Management System (FIMS). To

guarantee that the system could satisfy the demands of high-security environments and provide trustworthy file integrity verification, these issues had to be resolved.

4.1. Efficient Handling of Large Files and Batch Processing

Ensuring that the FIMS could process large files and manage batch operations was a primary challenge. In real-world applications, the system may encounter directories with numerous large files, and inefficient processing could lead to high memory usage, system slowdowns, or delays. This limitation would reduce the system's scalability and effectiveness in environments that require processing significant data volumes.

- **Solution**: To address this challenge, Java's java.io.File and java.nio.file.Files libraries were used to optimize file handling processes. These libraries provided efficient methods for reading and writing large files, which ensured that memory management was effective even during high-volume processing. Additionally, batch operations were streamlined to process directories efficiently, allowing the FIMS to handle numerous files without performance degradation. This optimization was validated through testing, confirming that the system maintains both speed and reliability across varying file sizes and volumes.

4.2. Implementing Secure Cryptographic Operations and Robust Error Handling

A dual challenge arose in ensuring secure cryptographic implementation and managing errors effectively. The use of HMAC-SHA3 for hashing required careful key management to maintain security, as hardcoding sensitive keys could expose the system to unauthorized access. Furthermore, robust error handling was necessary to manage unexpected conditions, such as invalid file paths or unsupported file formats, without compromising the system's stability or exposing sensitive information.

- **Solution**: The javax.crypto library was used to securely generate and manage HMAC-SHA3 hashes. To protect the HMAC key, it was stored separately in a secure configuration file or environment variable, which was accessible only to authorized personnel. This approach ensured that sensitive data was not embedded in the source code, reducing the risk of key exposure. For error handling, comprehensive exception management was implemented, covering common errors such as IOException and InvalidKeyException. Informative yet secure error messages were added to assist users without disclosing sensitive information. This combined approach reinforced both the security and reliability of the FIMS, enabling it to operate consistently and securely, even in error-prone conditions.

5. Conclusion

In settings where data authenticity is crucial, the File Integrity Management System (FIMS) created for this project offers a strong and dependable way to protect file integrity. The system's use of HMAC-SHA3-256 guarantees safe hash generation and verification, enabling businesses to efficiently identify unwanted file changes. The two applications, CreateHash and VerifyHash, provide a scalable and impenetrable approach to data security by enabling the smooth generation, storing, and verification of cryptographic hashes.

The FIMS proved to be accurate, efficient, and secure during its deployment, proving that it is appropriate for high-security industries like finance, healthcare, and transportation infrastructure. The system's ability to reliably identify even minute file changes and manage massive data volumes without experiencing performance issues was confirmed by extensive testing. Comprehensive security features, such as collision-resistant hashing, secure key management, and strong error handling, also increased the system's resistance to unwanted access and manipulation attempts.

In addition to being a useful tool for businesses that value data integrity, the FIMS offers a useful framework for adhering to legal requirements that demand safe data handling procedures. The FIMS improves operational stability, facilitates regulatory compliance, and strengthens the overall cybersecurity posture of businesses handling sensitive data with its dependable file verification procedure.

In summary, the FIMS provides an efficient method for managing file integrity while addressing the crucial requirement for data authenticity in digital contexts. This system may easily be modified to satisfy the changing needs of cybersecurity in high-stakes industries and lays a solid basis for future improvements, such as possible integration with real-time monitoring systems.

Compliance with ethical standards

Disclosure of conflict of interest

No conflict of interest to be disclosed.

References

[1] National Institute of Standards and Technology. (2014). Federal Information Processing Standards Publication: SHA-3 Standard. Retrieved from [https://csrc.nist.gov/files/pubs/fips/180-4/upd1/final/docs/fips180-4-draft-aug2014.pdf]

[2] Oracle. (2018). Java Cryptography Architecture (JCA) Reference Guide. Retrieved from [https://docs.oracle.com/javase/8/docs/technotes/guides/security/crypto/CryptoSpec.html]

[3] Oracle. (2023). Java Platform, Standard Edition Documentation, Version 8 API Specification. Retrieved from [https://docs.oracle.com/javase/8/docs/api/]

Appendix: Code for File Integrity Management System

CreateHash.java

```
import java.io.File;

import java.io.IOException;

import java.nio.file.Files;

import javax.crypto.Mac;

import javax.crypto.spec.SecretKeySpec;

import java.security.NoSuchAlgorithmException;

public class CreateHash {

 public static void main(String[] args) throws Exception {

 File sourceDir = new File(args[0]);

 File destinationDir = new File(args[1]);

 for (File file : sourceDir.listFiles()) {

 byte[] content = Files.readAllBytes(file.toPath());

 byte[] hash = generateHMACSHA3(content);

 Files.write(new File(destinationDir, file.getName() + ".hash").toPath(), hash);

 }

 }

 public static byte[] generateHMACSHA3(byte[] content) throws NoSuchAlgorithmException {

 Mac hmacSha3 = Mac.getInstance("HmacSHA3-256");

 SecretKeySpec keySpec = new SecretKeySpec("secret-key".getBytes(), "HmacSHA3-256");
```

```
 hmacSha3.init(keySpec);
 return hmacSha3.doFinal(content);
 }
}
```

VerifyHash.java

```
import java.io.File;
import java.nio.file.Files;
import java.util.Arrays;
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.security.NoSuchAlgorithmException;
public class VerifyHash {
 public static void main(String[] args) throws Exception {
 File sourceDir = new File(args[0]);
 File destinationDir = new File(args[1]);
 for (File file : sourceDir.listFiles()) {
 byte[] content = Files.readAllBytes(file.toPath());
 byte[] storedHash = Files.readAllBytes(new File(destinationDir, file.getName() + ".hash").toPath());
 byte[] currentHash = generateHMACSHA3(content);
 System.out.println(file.getName() + (Arrays.equals(storedHash, currentHash) ? " YES" : " NO"));
 }
 }
 public static byte[] generateHMACSHA3(byte[] content) throws NoSuchAlgorithmException {
 Mac hmacSha3 = Mac.getInstance("HmacSHA3-256");
 SecretKeySpec keySpec = new SecretKeySpec("secret-key".getBytes(), "HmacSHA3-256");
 hmacSha3.init(keySpec);
 return hmacSha3.doFinal(content);
 }
}
```