Kiran Shridhar Alva

University of Pittsburgh

Independent Study Project Report

Professor Balaji Palaniswamy

# Blockchain-based File Integrity Verification in HDFS

Table of Contents

[Introduction 2](#_Toc184047025)

[System Requirements 7](#_Toc184047026)

[Architecture 8](#_Toc184047027)

[Setup and Installation 9](#_Toc184047028)

[Implementation Details 14](#_Toc184047029)

[Testing and Validation 22](#_Toc184047030)

[Conclusion 24](#_Toc184047031)

[References 25](#_Toc184047032)

Project Report: Blockchain-based File Integrity Verification in HDFS

# Introduction

Overview of the Project

This project implements a **Blockchain-Based File Integrity Verification System** that integrates **Hadoop Distributed File System (HDFS)** and **Ethereum Blockchain** to ensure the integrity of files and directories. It addresses the need to detect tampering or malicious modifications to files stored in a distributed environment. By leveraging blockchain's immutability, the system tracks file metadata and directory-level hashes to detect unauthorized changes.

**Key Features:**

1. **File Metadata Hashing:**
   * The system uses the **SHA-256** algorithm to generate hashes for the metadata of files stored in HDFS.
   * These hashes uniquely represent the state of a file's metadata and ensure consistency.
2. **Blockchain Integration:**
   * An Ethereum **smart contract** manages and stores file and directory metadata hashes.
   * The store Hash function records hashes on the blockchain, ensuring tamper-proof storage.
   * The get Hash function retrieves stored hashes for comparison during integrity verification.
3. **Merkle Root for Directories:**
   * Generates a **Merkle root hash** for an entire directory by combining metadata hashes of individual files.
   * Provides an efficient and hierarchical way to verify directory integrity.
4. **Tampering Detection:**
   * Compares the current metadata hash of a file or directory with the blockchain-stored hash to detect discrepancies.
   * Identifies unauthorized modifications, deletions, or additions of files.
5. **Malicious Attempt Simulation:**
   * Simulates malicious actions like adding a new file or modifying existing file content to test the system's ability to detect tampering.
6. **Tampered File Identification:**
   * Scans files in a directory to identify which specific files have been tampered with.
7. **HDFS Integration:**
   * Utilizes the **InsecureClient** library for HDFS operations such as listing directories, retrieving file metadata, and writing to files.

**Core Components:**

1. **Smart Contract:**
   * Written in Solidity, it serves as a decentralized ledger for storing and retrieving file and directory hashes.
   * Deployed using the Web3.py library on a local Ethereum network.
2. **Hashing Mechanism:**
   * File metadata is converted to JSON and hashed using the **SHA-256** algorithm.
   * For directories, a Merkle tree approach computes a root hash from individual file hashes.
3. **HDFS Client:**
   * Interacts with the Hadoop Distributed File System to manage files and retrieve metadata.
4. **Blockchain Integration:**
   * Ethereum smart contracts store hashes to prevent tampering and provide immutable verification.

**Use Case Workflow:**

1. **Initialization:**
   * Create a directory in HDFS and ensure its existence.
   * Generate SHA-256 hashes for each file's metadata and store them in the blockchain.
2. **Directory Hash Storage:**
   * Compute the Merkle root hash for the directory and store it in the blockchain.
3. **Tampering Simulation:**
   * Simulate malicious activity by adding or modifying files in the directory.
4. **Detection:**
   * Compare current hashes with blockchain-stored hashes to identify tampered files or directories.
   * Detect unauthorized additions, deletions, or modifications at the directory level using Merkle root comparison.
5. **Integrity Assurance:**
   * Verify the integrity of files and directories, ensuring secure and tamper-proof storage.

**Applications:**

1. **Data Integrity in Distributed Systems:**
   * Ensures data integrity in large-scale distributed environments like Hadoop clusters.
2. **Tamper Detection:**
   * Protects sensitive data against unauthorized changes, ensuring compliance with data security standards.
3. **Blockchain-Powered Audit Trails:**
   * Provides an immutable, blockchain-based record of file state for auditing and regulatory purposes.
4. **Secure File Storage Systems:**
   * Ideal for applications requiring secure storage and frequent integrity checks, such as healthcare, finance, or government records.

# Architecture

**System Design**

A diagram of a diagram

Description automatically generated

# **Objective**

# To design and implement a system that:

# Tracks file metadata in a secure blockchain environment.

# Detects unauthorized changes in files and the directory stored in HDFS.

# **Key Features**

# Secure storage of file metadata hashes on the Ethereum blockchain.

# Real-time detection of tampered files in HDFS.

# Simulated malicious attempts to test the robustness of the system.

# System Requirements

# **Hardware**

# Minimum: 8 GB RAM, Dual-core processor.

# Recommended: 16 GB RAM, Quad-core processor.

# **Software**

# Hadoop 3.4

# Python 3.9 with required libraries (web3, hdfs, solcx): Anaconda Navigator and VS Code

# Ethereum node (Ganache or Geth).

# Solidity 0.8.19.

# **Tools and Libraries**

# **Blockchain**: Web3.py for Ethereum interactions.

# **Distributed Storage**: HDFS client for file operations.

# **Hashing**: SHA-256 for metadata integrity checks.

# Setup and Installation

**Coding Environment Setup using VS Code and Anaconda Navigator**

1: Install Conda

1. Anaconda (full distribution) Download Conda:
   * Visit [Anaconda](https://www.anaconda.com/) for the full version
2. Install Conda:
   * Follow the installation wizard for your operating system (Windows, macOS, Linux).
3. Verify Installation:
   * Open a terminal and run: conda --version
   * You should see the version of Conda installed.

2: Install Visual Studio Code

1. Download and install Visual Studio Code from <https://code.visualstudio.com/>.
2. Open VSCode after installation.

3: Install Python Extension in VSCode

1. Open VSCode and navigate to the Extensions view:
   * Press Ctrl + Shift + X (Windows/Linux) or Cmd + Shift + X (Mac).
2. Search for "Python" and install the extension by Microsoft.

4: Open command prompt in VS code navigate to the path you want the Virtual Env installed and run the command : conda create -p venv python ==3.9 -y

5: Now activate the environment: conda activate venv/

A black background with white text

Description automatically generated

6: Install the required libraries

* pip install web3
* pip install py-solc-x
* pip install hdfs

**Ethereum Setup**

Step 1: Install Node.js and npm in venv folder.

Ganache CLI requires Node.js, which includes npm (Node Package Manager).

1. Download Node.js:
   * Visit <https://nodejs.org/> and download the LTS version for your operating system.
2. Install Node.js:
   * Follow the installation wizard for your operating system.
3. Verify the installation:
   * Open a terminal in and run:
     1. **node -v**
     2. **npm -v**

Step 2: Install Ganache CLI

1. Install Ganache CLI globally using npm : **npm install -g ganache**
   * This installs Ganache CLI globally on your system.
2. Verify the installation:
   * Run the following command: **ganache –version**

Step 3: Start Ganache CLI

1. Navigate to your project directory: cd K:\HDFS\_BLOCKCHAIN
2. Start Ganache CLI: ganache

A screenshot of a computer

Description automatically generated

A black screen with yellow text

Description automatically generated

**Hadoop Setup**

1. Download Hadoop:
   * Visit the official Apache Hadoop website to download Hadoop 3.2.4
2. Install Java Development Kit (JDK):
   * Hadoop requires Java to run. java version "1.8.0\_421"
   * Download and install the appropriate JDK version for your system.
3. Set Environment Variables:
   * JAVA\_HOME:
     + Navigate to System Properties > Environment Variables.
     + Create a new system variable named JAVA\_HOME and set its value to the path of your JDK installation.
   * HADOOP\_HOME:
     + Create another system variable named HADOOP\_HOME and set its value to the path of your Hadoop directory.
   * Update PATH Variable:
     + Append %JAVA\_HOME%\bin and %HADOOP\_HOME%\bin to the system PATH variable.
4. Configure Hadoop:
   * Navigate to the Hadoop K:\hadoop-3.2.4\hadoop-3.2.4\
   * Create a data directory
   * Create a namenode and datanode directory
   * Add these file paths in the config files

Configuration

Core-site

A computer screen shot of a computer code

Description automatically generated

Hdfs-site

A computer screen shot of a computer code

Description automatically generated

Yarn-site

A screenshot of a computer program

Description automatically generated

Format the Namenode:

* + Open Command Prompt and execute: hdfs namenode -format

1. Start Hadoop Services:
   * Navigate to the Hadoop sbin directory.
   * Start the NameNode and DataNode: start-dfs.cmd
   * Start the ResourceManager and NodeManager: start-yarn.cmd
   * Type jps and you should see this

A black screen with white text

Description automatically generated

# Implementation Details

Test Files

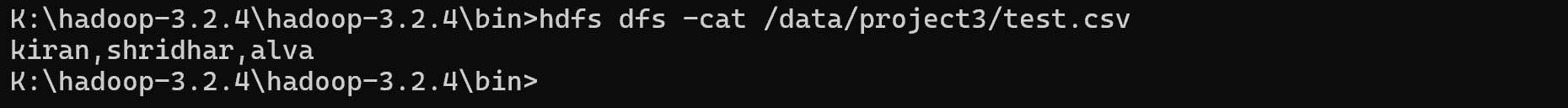
A screenshot of a computer

Description automatically generated

Store test files of different formats (csv,json,txt) in a directory in hadoop

A screen shot of a computer program

Description automatically generated



A computer screen with white text

Description automatically generated

Code Structure and Modules

**Initialization**

A computer screen with text on it

Description automatically generated

**Metadata Hash Generation**:

A screen shot of a computer program

Description automatically generated

**Smart Contract Deployment**:

A screen shot of a computer program

Description automatically generated

**Storing and retrieving hash from blockchain**

A screen shot of a computer code

Description automatically generated

**Tamper Detection**:

A computer screen shot of a program code

Description automatically generated

**Directory level Hash Generation using merkle tree**

A screen shot of a computer code

Description automatically generated

**Directory Level Integrity Check**

A screen shot of a computer program

Description automatically generated

**Malicious Attempt Simulation**

A screen shot of a computer code

Description automatically generated

**Execution**

**A screen shot of a computer program

Description automatically generated**

Explanation of Key Algorithms/Functions

The project integrates **Hadoop Distributed File System (HDFS)** and **Ethereum Blockchain** to create a secure and efficient mechanism for verifying file and directory integrity. It ensures that any unauthorized modifications, such as tampering, addition, or deletion of files, are reliably detected. The core functionality relies on generating cryptographic hashes for file metadata and leveraging blockchain's immutability for secure storage and verification.

The process begins with the generate\_metadata\_hash function, which retrieves a file's metadata from HDFS using the HDFS client. The metadata is serialized into a JSON string, sorted to maintain consistency, and then hashed using the **SHA-256** algorithm. This hash uniquely represents the current state of the file's metadata. The hash is stored securely in the Ethereum blockchain using the store\_hash\_in\_blockchain function. This function interacts with a Solidity smart contract deployed on the blockchain, where the hash is permanently recorded alongside the file path using the storeHash method. Blockchain's immutability ensures that the stored hash cannot be tampered with.

To verify file integrity, the get\_hash\_from\_blockchain function retrieves the previously stored hash from the blockchain for a specific file path. The detect\_tampered\_files function iterates through all files in a specified HDFS directory, generating a current metadata hash for each file and comparing it with the blockchain-stored hash. If the two hashes differ, the file is flagged as tampered, indicating potential unauthorized modifications.

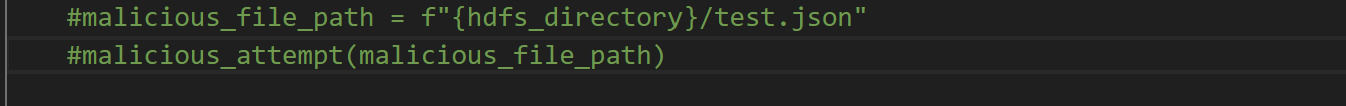
For directory-level integrity verification, the project uses a Merkle tree structure to compute a Merkle root hash. The generate\_directory\_hash function generates metadata hashes for all files in a directory and combines them iteratively using the compute\_merkle\_root function. This function pairs the hashes, hashes them together, and repeats the process until a single Merkle root hash remains, representing the integrity of the entire directory. The Merkle root is stored on the blockchain using the store\_directory\_hash\_in\_blockchain function. The detect\_directory\_tampering function compares the stored directory hash with the current Merkle root hash to detect tampering, such as file additions, deletions, or modifications.

The project also includes functionality to simulate malicious activities, such as adding or modifying files in HDFS, to test the system's ability to detect tampering. This integration of HDFS and blockchain combines the scalability and performance of centralized storage with the security and transparency of decentralized systems, offering a robust and efficient solution for ensuring data integrity in distributed environments.

# Testing and Validation

Test Cases and Results

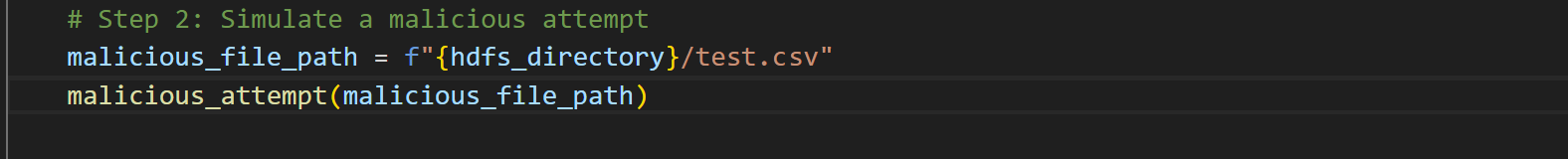
No malicious attempt

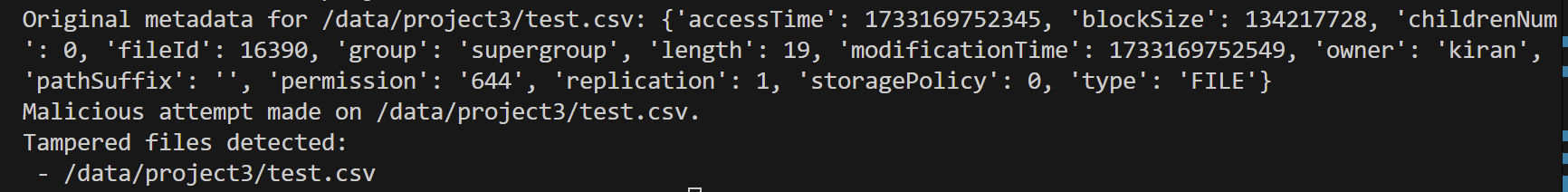


A screen shot of a computer program

Description automatically generated

Malicious attempt on csv file







Malicious attempt on JSON file

A screen shot of a computer code

Description automatically generated

A black screen with white text

Description automatically generated

Directory level integrity check

A screen shot of a computer program

Description automatically generated

# Conclusion

Blockchain serves as an efficient integrity verification system for HDFS, leveraging the strengths of both decentralized and centralized systems. By combining blockchain's immutability and transparency with HDFS's scalability and storage efficiency, this approach ensures robust data integrity and security. It addresses the shortcomings of each system individually providing the immutability and trust of decentralization while maintaining the high performance and centralized control of HDFS. This integration creates a reliable, tamper-proof, and scalable solution for verifying file integrity in distributed environments.

# References

Research Papers and Articles

* <https://www.sciencedirect.com/science/article/pii/S2096720921000270#:~:text=In%20BlockHDFS%2C%20the%20blockchain%20is,of%20a%20file%20from%20HDFS>.
* University of Pittsburgh INFSCI 2750 Mini Project 3 Blockchain-assisted Verifiable Cassandra.
* Ethereum white paper. <https://ethereum.org/en/whitepaper/>.
* Merkle tree. <https://en.wikipedia.org/wiki/Merkle_tree> .

Tools and Libraries Documentation

* Hadoop Installation : <https://www.youtube.com/watch?v=knAS0w-jiUk>
* <https://www.oracle.com/java/technologies/javase/javase8-archive-downloads.html>

Acknowledgments

I extend my deepest gratitude to **Professor Balaji Palanisamy** of the School of Computing and Information at the University of Pittsburgh for his invaluable insights and expertise. His guidance in the fields of distributed systems and blockchain technology has significantly contributed to the successful completion of this project.