Report

Group Members:

- Bhavana Devulapally
- Shusrita Venugopal
- Neha Navarkar

Project Description:

The project entails developing a proof-of-concept system consisting of several integral components. The Data Owner (DO) is tasked with preparing a set of key-value data and constructing a local Merkle Hash Tree (MHT) over this data. This is achieved through the creation of a class `DataOwner` with methods `prepare_data(data)` and `build_mht(data)`. The Database Service Provider (SP) operates as a server hosting a Cassandra database, responsible for interacting with Cassandra to store data received from the DO. This functionality is implemented through a class `DatabaseServiceProvider` with methods `store_data(data)` and `receive_data(data)`. The Query Client (C) issues key-value queries to SP, supporting query issuance and result verification utilizing the MHT, implemented in a class `QueryClient` with methods `issue_query(key)` and `verify_query_result(key, result, mht_root)`. Additionally, a Malicious Client (MC) is designed to serve as an adversary, tampering with data stored in Cassandra via a class `MaliciousClient` with the method `tamper_data(data)`. The Ethereum Blockchain component is not to be implemented, with relevant codes provided externally. This system aims to demonstrate secure data storage and retrieval mechanisms with integrity verification in a distributed environment.

Verification Results:

1. We decided to use below key value data set for MT implementations in Python.

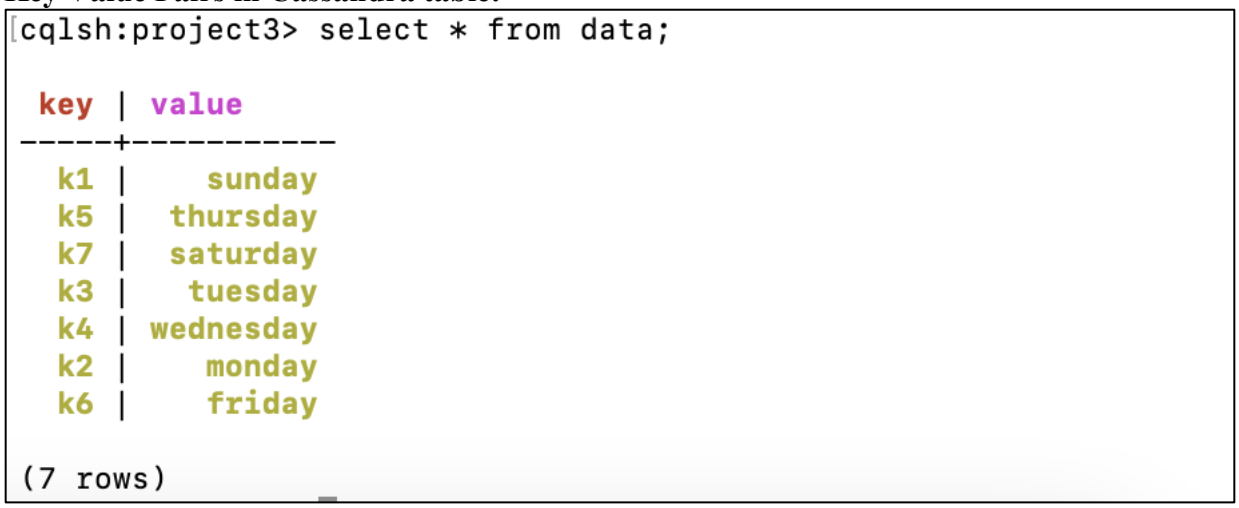
```
ubuntu@group6:~/project3$ python3 driver.py
k1:sunday
k2:monday
k3:tuesday
k4:wednesday
k5:thursday
k6:friday
k7:saturday
```

2. Attack Verification Results:

i. No Attack:

In this scenario, only a query is executed, and there is no involvement of a Malicious Client (MC) attempting to serve as an adversary to manipulate data stored in Cassandra.

Key Value Pairs in Cassandra table:



Observation:

In this scenario, we have three main actors: the client (C), the server provider (SP), and a Merkle hash tree (MHT). The client, represented by C, interacts with the server provider (SP) to issue query requests and retrieves data. Additionally, C has the capability to verify the integrity of the data received from SP using a Merkle hash tree (MHT) implemented on the client side.

Here's how the process typically works:

1. Query Requests from Client (C): C initiates query requests to the server provider (SP) to retrieve specific data or perform certain operations on the data stored in the database.

2. Data Retrieval from SP: Upon receiving the query requests, the server provider (SP) executes the queries on the database and retrieves the requested data.

3. Data Verification by Client (C): Once C receives the data from SP, it verifies the integrity of the data using a Merkle hash tree (MHT) implemented on its own side. This involves generating a Merkle root hash from the received data and comparing it with the Merkle root hash stored on C's side.

4. Integrity Verification with MHT: C computes the Merkle root hash of the received data and compares it with the Merkle root hash stored on its side. If the two hashes match, it indicates that the received data is intact and has not been tampered with during transmission. However, if the hashes do not match, it suggests that the data may have been modified or corrupted.

ii. Attach has happened:

In this scenario, we initiated the Malicious Client (MC) and subsequently conducted queries, along with verification, to demonstrate the presence of tampered data within Cassandra.

Key Value Pairs in Cassandra table:

```
cqlsh:project3> select * from data;

 key | value
-----+-----------
  k1 |  february
  k5 |  thursday
  k7 |  saturday
  k3 |   tuesday
  k4 | wednesday
  k2 |    monday
  k6 |    friday

(7 rows)
```

Observations:

In this scenario, we have a malicious client (MC) acting as an adversary to tamper with data stored in the Cassandra database running on the server provider (SP). Additionally, blockchain technology is employed to detect whether the data has been tampered with or not.
1. Malicious Client (MC): MC is an entity with malicious intent, aiming to compromise the integrity of the data stored in the Cassandra database hosted by the server provider (SP). MC may attempt to alter or manipulate the data stored in the database to achieve its malicious objectives.

2. Server Provider (SP): SP operates the Cassandra database, which stores sensitive or critical data. The database serves as a repository for various types of information, such as user data, financial records, or transactional data. SP is responsible for ensuring the security and integrity of the data stored in the Cassandra database.

3. Blockchain Technology: Blockchain technology is leveraged as a tamper-evident and immutable ledger to detect any unauthorized modifications or tampering of the data stored in the Cassandra database. By storing cryptographic hashes of the data records or transactions on the blockchain, any changes made to the data can be detected and verified through consensus mechanisms inherent in blockchain networks.

4. Detecting Data Tampering: Whenever MC attempts to tamper with the data stored in the Cassandra database, the changes made to the data will be reflected in the cryptographic hashes stored on the blockchain. The blockchain network, through its consensus mechanism, will detect the discrepancies between the original hashes and the tampered hashes, signaling the presence of data tampering.

By combining the use of Cassandra for data storage, blockchain for tamper detection, and proactive security measures, the system can effectively detect and respond to malicious attempts to tamper with the data, safeguarding the integrity and trustworthiness of the stored information.

Code Explanation:
1. Malicious Client (MC) Behavior: The malicious client (MC) modifies the data stored in the Cassandra database by changing the value associated with the key "k1" to "february".

2. Data Retrieval and Verification: After the data modification, the client (C) issues a query to retrieve the data associated with the key "k1". It then computes the hash of the retrieved value and compares it with the Merkle proof obtained from the blockchain.

3. Output Interpretation:
- Tampered Value: The value retrieved from the Cassandra database has been tampered with and changed to "february".
- Hash Value: The hash value computed for the tampered data is "69501faaf3fabf7df57b18a05c8c6a5aa6c3fb7fc3ec8781f5b2390367ee865a".
- Proofs of the Key: The Merkle proofs obtained from the blockchain for the key "k1" are provided.
- Merkle Root: The Merkle root hash of the Merkle tree stored in the blockchain is provided.
- Verification Result: The retrieved value is determined to be modified based on the comparison between the computed hash and the Merkle proofs.

Conclusion:
The output indicates that the retrieved value has been tampered with, as the computed hash value does not match the Merkle proofs obtained from the blockchain. This demonstrates the effectiveness of using Merkle trees for detecting data tampering and ensuring data integrity in distributed systems.

3. Blockchain Ganache service in 2nd terminal

i. Genesis Block

```
HD Wallet
==================
Mnemonic:      stage digital gold blue solution tobacco dove rigid market equip enemy scene
Base HD Path:  m/44'/60'/0'/0/{account_index}

Default Gas Price
==================
2000000000

BlockGas Limit
==================
30000000

Call Gas Limit
==================
50000000

Chain
==================
Hardfork: shanghai
Id:       1337

RPC Listening on 127.0.0.1:8545
eth_accounts
eth_getBlockByNumber
eth_chainId
eth_chainId
eth_estimateGas
eth_blockNumber
eth_getBlockByNumber
eth_sendTransaction

  Transaction: 0x4ae2d745cf77917ce579526ffb0bb286cfa5736a6e50cf3b8a924fbcbb51bfaf
  Contract created: 0x58aae3f29990532bf7461def3558a9f1c82d29cd
  Gas usage: 309939
  Block number: 1
  Block time: Tue Apr 23 2024 17:07:12 GMT+0000 (Coordinated Universal Time)
```

ii. Followed by two transaction blocks (Attack and No Attack)

```
eth_getTransactionReceipt
eth_getBlockByNumber
eth_chainId
eth_chainId
eth_estimateGas
eth_blockNumber
eth_getBlockByNumber
eth_sendTransaction

  Transaction: 0xc27aa58a078551bdbd2c4c99293b8f1a12ec0eb402b49ac0a728982e7e336768
  Gas usage: 90222
  Block number: 2
  Block time: Tue Apr 23 2024 17:07:12 GMT+0000 (Coordinated Universal Time)

eth_getTransactionReceipt
eth_chainId
eth_call
eth_chainId
eth_call
eth_chainId
eth_call
eth_accounts
eth_getBlockByNumber
eth_chainId
eth_chainId
eth_estimateGas
eth_blockNumber
eth_getBlockByNumber
eth_sendTransaction

  Transaction: 0x0cad6f749e73cdb2c0e555989a590e457511442103b22492c39714b1e937facd
  Contract created: 0x33e780d6a9c7d2a2b66f613ca0b5399074ddfbcf
  Gas usage: 309939
  Block number: 3
  Block time: Tue Apr 23 2024 17:13:15 GMT+0000 (Coordinated Universal Time)
```

```
eth_getTransactionReceipt
eth_getBlockByNumber
eth_chainId
eth_chainId
eth_estimateGas
eth_blockNumber
eth_getBlockByNumber
eth_sendTransaction

  Transaction: 0x7e0d931d063a5ada1463404fcf4dd9ab69298645dd2c41e4214f0556abc40a9a
  Gas usage: 90222
  Block number: 4
  Block time: Tue Apr 23 2024 17:13:15 GMT+0000 (Coordinated Universal Time)

eth_getTransactionReceipt
eth_chainId
eth_call
eth_accounts
eth_getBlockByNumber
eth_chainId
eth_chainId
eth_estimateGas
eth_blockNumber
eth_getBlockByNumber
eth_sendTransaction

  Transaction: 0x26ba89e1312ba445a4d2eb1a06f60781ada80a08a1fc3dfc6cdd03feedc3fd16
  Contract created: 0x34d3164a523dab275f614109f1a83b0523f66a8f
  Gas usage: 309939
  Block number: 5
  Block time: Tue Apr 23 2024 17:27:36 GMT+0000 (Coordinated Universal Time)
```