INFSCI 2750 Mini Project 3

Blockchain-based Cloud Data Verification

1 Project Introduction

Outsourced data service has been a classic paradigm in the domain of cloud computing, in which a data owner can delegate his/her own data to an outsourced cloud service provider to enjoy a powerful cloud service with a reasonable economic cost. Practically, Outsourced Database Model (*ODB*) is one of the real-world paradigms to realize an outsourced data service. In general, such a paradigm consists of the following key entities: *Data Owners (DOs)*, *Database Service Provider (SP, a server)*, and *Clients (Cs)*. From a high-level perspective, the workflow of *ODB* can be described as follows : A *DO* first creates his/her own database with an authenticated data structure over the created database and uploads them to an *SP*. Any client *C* can request a query with the *SP* to get related results from the database.

Such a service paradigm, however, inevitably exposures to the risk of an untrusted *SP*. Hence, on the side of clients, how to authenticate queried data from *SP* is of great importance. Fortunately, incorporating *Authenticated Data Structures (ADS)* with blockchain techniques shows a promising solution to achieve query authentications. In this project, we will focus on a simple construction following such a solution, which consists of the construction of *Merkle Hash Tree (MHT)*, serving as an ADS, and the adoption of smart contracts in the *Ethereum* blockchain. Next, we will show the holistic workflow of this project in Figure 1: (1) *DO* first creates his/her database and constructs an MHT based on the database. (2) *DO* then uploads the database and the built MHT to *SP* and records the Merkle root on *Ethereum*.(3) A query client *C* can request a query result from *SP*. (4)The *C* then gets the corresponding query result as well as the Merkle proof associated with the result.(5) *C* then retrieves the Merkle root in *Ethereum*. (6) Finally, *C* reconstructs an Merkle root and compare the value with the one retrieved from *Ethereum*. A detailed introduction of MHT will be provided as tutorial.

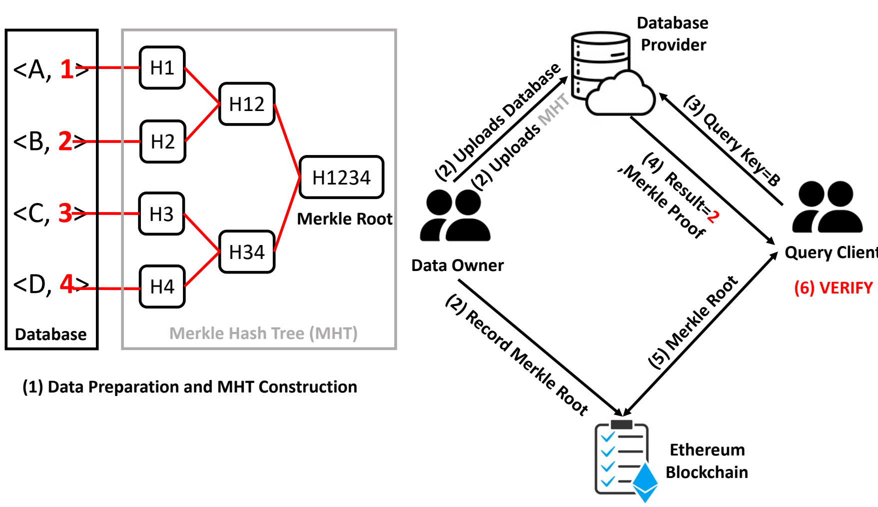

Figure 1: Blockchain-based Data Verification

In this project, you will gain some experiences to learn such a design with more detail in a simulated *proof-of-concept* environment. The detailed requirements will be discussed next.

2 Requirements

Environment Setup

1. You will need a *Python* environment (version $\geq$ 3.6.0) and the following packages in *Python*:
   - Web3.py
   - pymerkletools
   - py-solc-x
2. You will need to configure the environment of Solidity.
3. *Ethereum*: You will need to set up a local Ethereum blockchain environment with the help of *Ganache*.

Please run the file , named *smartcontract.sh*, to install all the configurations.

Data Preparation

- You will need to prepare your own key-value data set or data object, (K, V), which will serve as the data that needs to be outsourced. Such a data can be as simple as the following one:

```
    ori_data={
            ’A’: ’1’,
            ’B’: ’2’,
            ’C’: ’3’,
            ’D’: ’4’,
            ’E’: ’5’,
            ’F’: ’6’
        }

```

  Also, you will need to prepare your own key-index data for further merkle tree operations. For example,

```
     key_index = {
        ’A’: 0,
        ’B’: 1,
        ’C’: 2,
        ’D’: 3,
        ’E’: 4,
        ’F’: 5
    }

```

Merkle Tree Construction

- You will need to build a MHT over your own data. Specifically, you will need to build a MHT based on V in your (K, V) pairs. You will just need to replace `data` with your own dataset.

```
    def build_merkle_tree(data):
        """ Build Merkle Tree over data"""
        mt = merkletools.MerkleTools(hash_type="md5")
        for k, v in data.items():
            mt.add_leaf(v, True)
        mt.make_tree()
    return mt

```

- After building a MHT, you can get the built merkle tree object, `merkle_tree`, and the merkle root, `merkle_root`

```
    #build merkle tree
    merkle_tree=build_merkle_tree(ori_data)
    #get merkle root
    merkle_root=merkle_tree.get_merkle_root()

```

- `merkle_root` will be submitted to *Ethereum* through executing the following code. In this part, you will only need to replace your own RPC server address in line 25.

```python
## Ethereum Blockchain Deployment & Interaction##
compiled_sol = compile_source(
    '''
    pragma solidity >0.5.0;
    contract Verify{
        string merkleRoot;

        function setMerkleRoot(string memory _merkleRoot) public {
            merkleRoot=_merkleRoot;
        }

        function getMerkleRoot()view public returns (string memory){
            return merkleRoot;
        }
    }
    ''',
    output_values=['abi', 'bin']
)

contract_id, contract_interface = compiled_sol.popitem()

bytecode = contract_interface['bin']
abi = contract_interface['abi']

w3 = Web3(Web3.HTTPProvider('Please replace your own RPC address'))
w3.eth.default_account = w3.eth.accounts[0]

Verify = w3.eth.contract(abi=abi, bytecode=bytecode)

deploy_tx_hash = Verify.constructor().transact()
deploy_tx_receipt = w3.eth.wait_for_transaction_receipt(deploy_tx_hash)

verify = w3.eth.contract(
    address=deploy_tx_receipt.contractAddress,
    abi=abi
)
# data owner set merkle root
set_tx_hash = verify.functions.setMerkleRoot(merkle_root).transact()
set_tx_recipient = w3.eth.wait_for_transaction_receipt(set_tx_hash)
```

Data Verification-Original Data

- You will need to retrieve the value V corresponding to your search key K from your **original data** to get your query result.

```python
#Query Client start querying, this part is for original data query, no malicious event
query_result=query_data_by_key('B',ori_data)
```

- You will need to get the merkle proof of your received query result from the built MHT by executing

```python
# get the index in the merkle tree corresponding to your search key
merkle_proof_index_ori=get_merkle_index_by_key('B',key_index)
# get the merkle proof of your query result
merkle_proof_ori=get_merkle_proof_by_index(merkle_tree,merkle_proof_index_ori)
```

- You will need to finish the verification of your received result by following the next steps. First, you will need to retrieve the original merkle root from *Ethereum* by executing

```python
#get original merkle root from Ethereum
root_from_chain = verify.functions.getMerkleRoot().call()
```

Then, you will need to get the hash value of your query result by executing

```python
#get the hash of query result
query_result_hash=get_value_hash(query_result)
```

Followed by executing the following code to perform verification

```
    #verification
    is_valid_ori = merkle_tree.validate_proof(merkle_proof_ori , query_result_hash ,
    root_from_chain)
```

Finally, you will need to show the value of the verification result , `is_valid_ori`, in the example.

Data Verification-Modified Data

- You will need to manually modify your data. Specifically, modify the value of a specific key pair. For example, the associated value of $'B'$ is changed to $'3'$ in this example.

```
     modified_data={
        'A': '1',
        'B': '3',#Modified Pair
        'C': '3',
        'D': '4',
        'E': '5',
        'F': '6'
    }
```

- You will need to retrieve the value corresponding to your search key from your **modified data** to get a query result. Specifically, query with the key K corresponding to your modified pair (K, V'), by executing the following code

```
    #query on your modified data
     query_result_mali=query_data_by_key('B',modified_data)
```

- You will need to get the merkle proof of your received query result by executing

```
    #query on your modified data
    query_result_mali=query_data_by_key('B',modified_data)
    merkle_proof_index_mali=get_merkle_index_by_key('B',key_index)
    merkle_proof_mali=get_merkle_proof_by_index(merkle_tree,merkle_proof_index_mali)
```

- You will need to retrieve the original merkle root from the *Ethereum* blockchain by executing

```
    #Query Client start querying, this part is for modified data,malicious event
    occured
    query_result_mali=query_data_by_key('B',modified_data)
    merkle_proof_index_mali=get_merkle_index_by_key('B',value_index)
    merkle_proof_mali=get_merkle_proof_by_index(merkle_tree,merkle_proof_index_mali)
```

- You will need to finish the verification of your received result over the modified data by following the next steps. First, executing the following code to calculate the hash value of your query result

```
    #calculate hash value
    query_result_hash_mali = get_value_hash(query_result_mali)
```

- Followed by executing the following code to finish the verification

```
    #verification for modified data
    is_valid_mali = merkle_tree.validate_proof(merkle_proof_mali,
    query_result_hash_mali, root_from_chain)
```

Finally, you will need to show the value of the verification result, `is_valid_mali`, in the example.

3 Submission

- You will need not to implement any code, all helper functions and libraries will be provided in the sample code named *demo.py*. For configurations, please refer to the file named *smartcontract.sh*.
- You will need to provide your own data set and show a verification result based on your original data by showing a screenshot.

- You will need to provide your own data set and show a verification result based on your modified data by showing a screenshot.
- You will need to submit the screenshot of your Ethereum environment. Specifically, you will need to submit your deployed contract address.