INFSCI 2750 Mini Project 3

Blockchain-assisted Verifiable Cassandra

1 Background

Outsourced data service has been a classic paradigm in the domain of cloud computing, in which a data owner can delegate his/her data to an outsourced cloud service provider to enjoy a powerful cloud service with a reasonable economic cost. Outsourced Database Model (*ODB*) is one of the practical 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)*. Roughly, *ODB* works as follows : A *DO* prepares his/her data set and uploads them to an *SP*. Later, 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*. On the side of query clients, therefore, 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 design philosophy, which consists of the adoption of *Merkle Tree (MHT)*, serving as an ADS, and that of smart contracts supported by the *Ethereum* blockchain. Next, we will show the overview this construction in Figure 1:

- **Step (1)** *DO* will first prepare his/her own data set that will be stored later in the Cassandra database provided by SP; meanwhile, based on his/her data set, *DO* will build a MHT locally.
- **Step (2)** *DO* then will upload the prepared data set as well as the built MHT to *SP*, and record the Merkle root on *Ethereum*. After performing such operations, SP will hold the uploaded MHT. Also, SP will store the received key value data set in the Cassandra database maintained by him/herself.
- **Step (3)** Later, a query client C will request a query result from *SP*.
- **Step (4)** After interacting with SP, C will get the corresponding query result as well as the corresponding Merkle proof.
- **Step (5)** C will retrieve the Merkle root recorded in *Ethereum*.
- **Step (6)** Finally, to perform query verifications, C will reconstruct a Merkle root and compare it with the one retrieved from *Ethereum*.

Since MHT and the Ethereum blockchain are at the centerpiece of the solution shown in Figure 1, next, we will unpack intuitions behind such notions with concrete examples.

2 Preliminaries

2.1 Merkle Tree (MHT)

We only provide informal descriptions here, for formal treatments of MHT, please check [2, 4]. In Figure 1, we assume that a set KV of key value data, where $KV = \{< A, 1 >, < B, 2 >, < C, 3 >, < D, 4 >\}$. Based on KV, we build a MHT, denoted as MHT_{KV}, as follows:

- (1) Given the value of each corresponding key, we first hash it and get the hash of each key, where $H1 = \mathcal{H}(1)$[1],$H2 = \mathcal{H}(2)$, $H3 = \mathcal{H}(3)$, and $H4 = \mathcal{H}(4)$. Such four values form the leaf nodes of MHT_{KV}
- (2) We then construct the entire MHT_{KV} in a bottom-up fashion as follows: By hashing the concatenation of $H1$ and $H2$, we get $H12$, where $H12 = \mathcal{H}(\mathrm{H1}\|\mathrm{H2})$ and $\|$ refers to concatenation. Similarly, $H34 = \mathcal{H}(\mathrm{H3}\|\mathrm{H4})$. Finally, deriving $H1234 = \mathcal{H}(\mathrm{H12}\|\mathrm{H34})$, which yields the merkle root of MHT_{KV}, denoted as r.

[1] $\mathcal{H}$ is called a collision-resistance hash function []

After constructing MHT_{KV} as above, later, in case of a query authentication regarding the value of the key A is issued, the following steps will start:

- Upon receiving such a query request, the holder of MHT_{KV} will deliver the corresponding value $V(A)$ of A as well as the Merkle proof π generated from MHT_{KV}, where $\pi = \{H2, H34\}$. Once having $V(A)$ and π, one will be able to reconstruct a new merkle root r' as follows:

$$r' = \mathcal{H}(\mathcal{H}(\mathcal{H}(V(A)) \parallel H2) \parallel H34) \tag{1}$$

Finally, the verification will work as follows:

- If $r' = r$, then $V(A) = 1$. This means the queried value was not tampered.
- If $r' \neq r$, then $V(A) \neq 1$. This means that the queried value was not the original one and some attacks occured.

2.2 The Ethereum Blockchain

In this construction, we will store the generated merkle root r on a smart contract running atop the Ethereum blockchain. One can treat this part as a black box in which a tamper-resistant context is provided. For more detailed introduction of Ethereum, please check [1, 3].

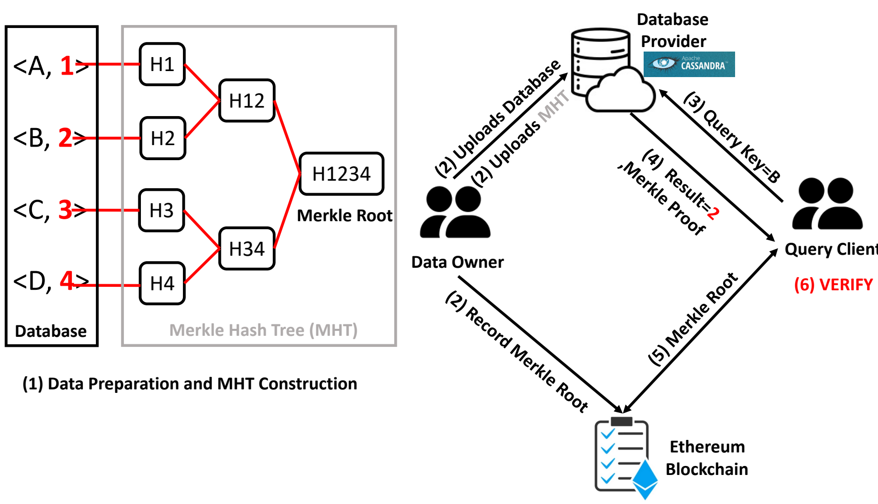

Figure 1: Blockchain-assisted Verifiable Cassandra

3 Project Description

3.1 Requirements

In this project, you will need to provide a *proof-of-concept* implementation sketched in Figure 1. To complete this project, your implementations have to, at least, consist of the following key classes:

- Data Owner (DO): DO takes charge of preparing a set of *keyvalue* data, building a local MHT over such data
- Database Service Prodiver (SP): SP serves as a server program running a Cassandra database. After getting the data from DO, SP will interact with the Cassandra to store such data in terms of a table.
- Ethereum Blockchain: You will not need to implement anything regarding this part. We will provide relevant codes.
- Query Client (C): C will be able to issue query requests to SP. In your implementation, you will need to support key value queries only. Also, C will be able to verify the resultant queries on his/her own side by adopting MHT.
- Malicious Client (MC): MC will serve as an adversary to tamper some data stored in the Cassandra running in SP.

3.2 Submissions

- You will need to prepare your own key value data set and submit all your implementations in *Python*.
- You will need to show the following two verification results:
  - **No Attack**: In this scenario, do not run your MC and show your verification result.
  - **Attack has happened**: In this scenario, you will need to run your MC first, followed by performing queries as well as verification, and showing your verification result.
- You will need to submit your table stored in your Cassandra by showing a screenshot.

References

[1] Ethereum white paper. https://ethereum.org/en/whitepaper/, [Online].

[2] Merkle tree. https://en.wikipedia.org/wiki/Merkle_tree, [Online].

[3] Andreas M Antonopoulos and Gavin Wood. *Mastering ethereum: building smart contracts and dapps.* O'reilly Media, 2018.

[4] Ralph C Merkle. A digital signature based on a conventional encryption function. In *Advances in Cryptology—CRYPTO'87: Proceedings 7*, pages 369–378. Springer, 1988.