IntegrityVault - A Secure Distributed Storage System

Kabilan Mahathevan[1]

[1]Department of Computer Science and Engineering, University of Moratuwa

July 25, 2024

1 Introduction

IntegrityVault is a secure distributed storage system that distributes file partitions as chunks across a distributed network. While ensuring data integrity, this system operates without incorporating fault tolerance mechanisms. IntegrityVault satisfies three basic requirements: it ensures the random distribution of file chunks among cluster nodes for security, allowing users to seamlessly retrieve and reconstruct the entire file at any time; it guarantees data integrity and validation by enabling verification of file chunks to confirm they have not been tampered with, using techniques such as a Merkle Tree; and it provides a user-friendly interface that allows users to list and search for stored files, view file metadata (including file name, size, date of storage, type, etc.), verify file integrity, download files, and upload new files to the distributed storage system.

A Merkle tree, is a data structure used in cryptography and computer science to ensure data integrity and efficiency. It is a binary tree in which every leaf node contains a hash of a data block, and each non-leaf node contains a hash of its child nodes. This hierarchical arrangement allows for quick and efficient verification of data integrity.

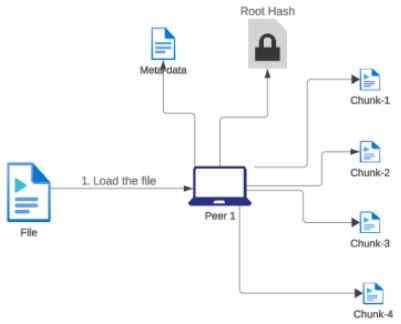

Figure 1: Partitioning the file into chunks

Figure 2: Sending File Metadata to the tracker

2 Methodology

Every IntegrityVault should have a centralized server called the *Tracker*. This Tracker server maintains all metadata information about files uploaded to IntegrityVault, allowing clients to view metadata before downloading files. This setup reduces the network call overhead between connected peers and clients in IntegrityVault, which would otherwise be necessary to obtain metadata information. Viewing available files is a frequent operation and could potentially overload the network if peers had to be queried for metadata information. Additionally, modifying metadata information on different peers can lead to file corruption even if the file chunks themselves are intact. To address these issues, the Tracker server stores the metadata information for uploaded files. An embedded database using *sqlite3* is utilized on the Tracker server to manage this metadata effectively.

When a client uploads a file, the file is first divided into equal-sized binary chunks, as specified in the configuration file as indicated in the Figure 1. The configuration file also includes the list of peers in the network and the size of the file chunks. After chunking the file, the chunks are randomly distributed among the specified peers

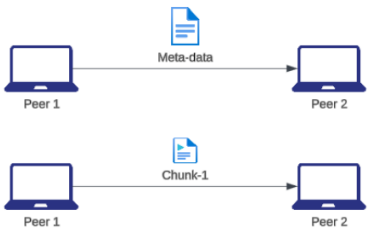

Figure 3: Distributing chunks to the Peers

in the network. This pseudo-random distribution helps reduce the overhead involved in checking peers with fewer available chunks, ensuring an even distribution. This method is particularly advantageous when scaling up to upload multiple files. Additionally, when files are uploaded, the Tracker server records all peers that have chunks of a particular file. This approach minimizes the need to flood the entire network to verify the existence of chunks on nodes. Since file chunks are randomly distributed across the storage network, metadata for each chunk is stored along with the chunk itself. This metadata is essential for retrieving and reconstructing the original file from its chunks, as the order of the chunks cannot be determined without it (Refer Figure 2, 3).

When a client wants to view an uploaded file, IntegrityVault fetches the metadata from the Tracker server. This metadata includes file information such as the file name, file size, and file type. The client is then presented with this information along with an option to download the file. Clicking the download button initiates a search for the file chunks in the storage network and retrieves them for the client.

When a client requests to download an uploaded file, the system first retrieves the list of peers that have chunks of the specified file from the Tracker server. It then fetches all the chunks from these peers along with their associated metadata. Using the metadata, IntegrityVault reconstructs the chunks to generate the complete file, which is then saved to the client's machine. Since IntegrityVault guarantees the integrity of uploaded files, it verifies whether the chunks have been tampered with during the reconstruction process. This ensures that the file remains intact and unaltered when it is reassembled from its chunks.

For integrity checks, IntegrityVault employs a Merkle tree. When a client uploads a file, it is divided into multiple chunks. A Merkle tree is then constructed in a bottom-up approach using these binary chunks, with SHA-256 used for hashing each chunk. The root hash value of the chunks is obtained from this process (Refer Figure 4). This root hash value is stored in the tracker server along with the file metadata. When a client requests to download a file, the chunks are retrieved from the distributed nodes, and a Merkle tree is constructed again from these chunks. This newly built Merkle tree is then compared with the root hash value stored in the Tracker server to ensure that the retrieved chunks have not been tampered with. Since it is mathematically impossible to tamper with a chunk in a way that matches the original root hash value, this process ensures the integrity of the distributed chunks.

3 Limitations and Future work

Since this is a proof-of-concept distributed storage system, it is designed to be used by only one client per Tracker server. To prevent redundant uploads, the file name is used as a unique key, and uploading a file with the same name will override the metadata of the existing file. Consequently, multiple clients using the same Tracker server can lead to data loss if they upload files with the same names. Additionally, since there is no authentication mechanism, multiple users connected to the Tracker server can download any client's files. For real-world usage, each client would need to manage their own Tracker server. Creating an authentication and authorization mechanism into IntegrityVault and mapping each user along with his own file in the tracker server can help to resolve this issue.

The distributed system assumes no network partition or node failures; hence, no recovery mechanism is implemented for node failures. Consequently, IntegrityVault does not include a mechanism for peers to connect to the network as storage nodes. The client must have a pre-defined list of IP addresses and ports of the distributed storage nodes to distribute the data. This is achieved through a configuration file on the client side that contains all the peers' information.

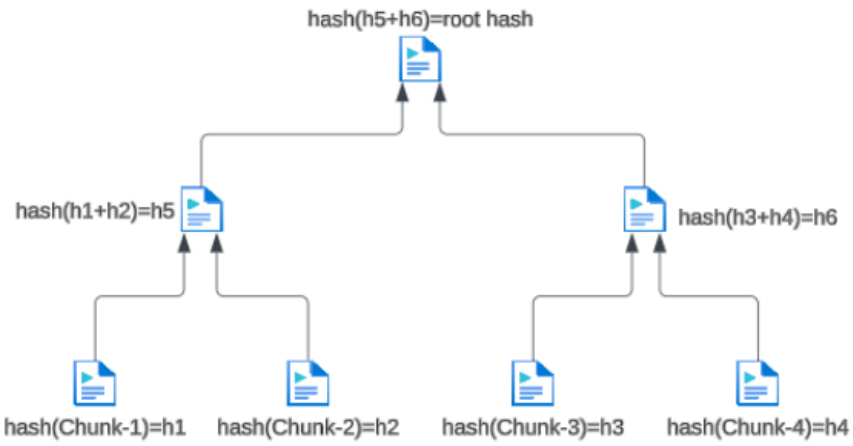

Figure 4: Merkel Tree built with the chunks