Coding Challenge - Merkle Tree

Valerio Mattioli

February 26, 2024

Contents

1 Introduction

The solution I developed is a monorepo composed of 3 subrepos:

1. The Merkle Tree library (libs/merkletree)
2. Server
3. Client

The client and the server both have their own Dockerfiles and a docker-compose to quickly run the demo just following the README.md.
Furthermore, I have setup also the Git Actions to build and test every time I push into "master" branch. (build.sh and test.sh are ran). I also have setup a githook to automatically format the files that I modified in the pre-commit

2 Library

While developing the project, I have started by building the merkle tree library (which was correctly working) while developing the test cases around it. In this way it is possible to safely improve the underlying algorithms while being sure to not break its core functionality.
Given that at the start of the week I was not at home I took advantage of this time to study Merkle Trees and tinker with a pen and piece of paper. For this first iteration of the algorithm I found myself more confident to represent the Merkle Tree as a matrix of hashes. The difficulty that I found is that the Merkle Tree is an inverted structure, it is built from the bottom-up: is not difficutl to build from the leaves but then is not trivial to gather the "merkle-proofs", given the inverted nature of the tree.
For this reason I opted for a matrix, in this way I can store once the constructed merkle tree and then I can quickly pick the right "merkle-proofs(i)" given the "file i" that I want to check the integrity.
So, then:

- the "leaf nodes" are the merkleTree[0]
- the "root hash" is the merkleTree[len(merkleTree)-1][0]

It is important to mention that, at this stage the Merkle Tree library is already well tested (>90% code coverage)

2.1 Functions

The library exposes 5 functions, specifically 2 used on the server and 3 on the client:

- ComputeMerkleTree - used by server
- ComputeMerkleProof - used by server
- ComputeRootHash - used by client
- ReconstructRootHash - used by client
- IsFileCorrect - used by client

Thanks to this test suite it is then possible to improve the performance of the algorithm very quickly while maintaining a correct workflow.
The library is using sha256 as hashing algorithm.

3 Server

For the scope of this demo, I'm storing the Merkle Tree in-memory and the uploaded files directly in server's filesystem. Nonetheless, I have already created the layers (handlers, services, repositories, models) for separating concerns to integrate persistency and S3/minio storage quickly when needed and already isolate the units to run tests properly per single unit.
To make the server production ready I put a rate limiter and the max file size and max batch upload size can be tweaked from the envvars as one can notice in the docker-compose.yml.
The server exposes a Swagger API at `http://localhost:8080/swagger/index.html`

4 Client

For the client I opted for a CLI application, which in my opinion is the best for the demo. Other than running the library's functions I also added a couple of flags (-dir and -store) to enable the user to define their own folders to use to bulk upload the files, download the file and store and retrieve the root-hash other than the default locations. In this way, the client is already quite user friendly with menus, helpers and descriptions. Given the limited time and that the client is only composed of 2 functions (upload and get) I used cobra to quickly develop and deliver the CLI.

5 Summary

I found the exercise very complete because of its wide technical coverage. I have particularly enjoyed getting into the rabbithole with the algorithm development. I like getting to the bottom of things, especially with cryptography (which is one of favorite subject!). I think that the applications are 100% compliant with the challenge request, with a high code coverage in the key modules. I am satisfied with the result and eager to discuss about it with the Recruiting and Technical Team. As a future development I would probably enable multi-clients (if needed), thread-safety and add a persistency layer, both on the data structures (DB) and on the files (S3, MinIO). Having the test suite already in place enables the development in parallel of both the library and the server to speed-up the delivery into production.