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.
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 a githook to automatically format the files that I modified in the pre-commit

2 Library

I will start this report from what concern the bulk of the code - the merkle tree library - is well tested (>90% code coverage).
I have given priority to first of all develop a basic library that works, then develop the test cases around it, in this way I can now 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, you build 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]

2.1 Functions

The library exposes 5 functions:

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

I'm happy with this first algorithm, for sure given more time I would explore deeper on how to implement it without using the matrix to represent the tree and retrieve the Proofs(i) efficiently. Now that I have a test suite I can improve the performance of the algorithm very quickly. What's important is that is working correctly.
The library is using sha256 as hashing algorithm.

3 Server

The server exposes a Swagger API and the handlers are tested. As a future development I would probably enable multi-clients, thread-safety and add a persistency layer, both on the data structures (DB) and on the files (S3, MinIO). For the scope of this demo I'm storing the merkle tree in-memory and the uploaded files directly in server's filesystem, but I already created the layers (handlers, services, repositories, models) for separating concerns to integrate persistency and S3 storage quickly when needed and already isolate units to unit tests properly. To make the server the most 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.

4 Client

For the client I opted for a CLI application, I think that this is the best way to deliver 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. So, the client - even though I developed it very quickly - 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 really had fun developing this challenge!
The exercise is very complete, because you have to touch a little bit of everything technically and then for the algorithm I really had fun getting into the rabbithole, is really something that I enjoy doing, I love cryptography! The applications are doing completely what the challenge is asking, and is doing that consistently with a high code coverage in the key modules. So, I am happy with the solution, I am really looking forward to discuss about it together.
As I said, if have to continue on the development of this application I would probably start by giving to the server persistency and the capacity to serve several clients (if is needed, of course), then in parallel I will look into improving the algorithm, 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.