



Dissertation on

“Layer 7 Integrity Check For Cloud Service Providers (L7ICCSP)”

Submitted in partial fulfilment of the requirements for the award of degree of

**Bachelor of Technology
in
Computer Science & Engineering**

UE17CS490A – Capstone Project Phase - 1

Submitted by:

| | |
|------------------------|----------------------|
| R Siva Girish | PES1201700159 |
| Gaurav C.G | PES1201700989 |
| Sangam Kedilaya | PES1201701139 |
| Zenkar Srinivas | PES1201701532 |

Under the guidance of

Prof. Prasad B Honnavalli

Professor, Computer Science and Engineering
Director, Centre for Information Security,
Forensics and Cyber Resilience (C-ISFCR)
Director, Centre for Internet of Things (C-IoT)

August - December 2020

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
FACULTY OF ENGINEERING
PES UNIVERSITY**

(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India



PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013)

100ft Ring Road, Bengaluru – 560 085, Karnataka, India

FACULTY OF ENGINEERING

CERTIFICATE

This is to certify that the dissertation entitled

‘Layer 7 Integrity Check For Cloud Service Providers(L7ICCSP)’

is a bonafide work carried out by

**R Siva Girish
Gaurav C.G
Sangam Kedilaya
Zenkar Srinivas**

**PES1201700159
PES1201700989
PES1201701139
PES1201701532**

in partial fulfilment for the completion of seventh-semester Capstone Project Phase - 1 (UE17CS490A) in the Program of Study - Bachelor of Technology in Computer Science and Engineering under rules and regulations of PES University, Bengaluru during the period Aug. 2020 – Dec. 2020. It is certified that all corrections/suggestions indicated for internal assessment have been incorporated in the report. The dissertation has been approved as it satisfies the 7th-semester academic requirements in respect of project work.

Signature
Prof. Prasad B Honnavalli
Professor

Signature
Dr. Shylaja S S
Chairperson

Signature
Dr. B K Keshavan
Dean of Faculty

External Viva

Name of the Examiners

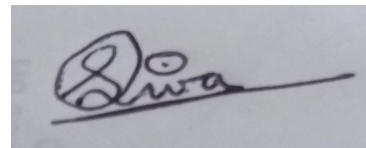
1. _____
2. _____

Signature with Date

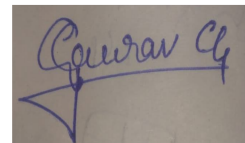
DECLARATION

We hereby declare that the Capstone Project Phase - 1 entitled “**Layer 7 Integrity Check For Cloud Service Providers(L7ICCSPP)**” has been carried out by us under the guidance of Prof. H B Prasad, Professor and submitted in partial fulfilment of the course requirements for the award of the degree of **Bachelor of Technology in Computer Science and Engineering of PES University, Bengaluru** during the academic semester August – December 2020. The matter embodied in this report has not been submitted to any other university or institution for the award of any degree.

PES1201700159 R Siva Girish



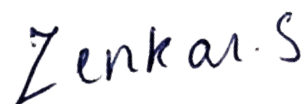
PES1201700989 Gaurav C.G



PES1201701139 Sangam Kedilaya



PES1201701532 Zenkar Srinivas



ACKNOWLEDGEMENT

I would like to express my gratitude to Prof. Prasad B Honnavalli, Department of Computer Science and Engineering, PES University, for his continuous guidance, assistance, and encouragement throughout the development of this UE17CS490A - Capstone Project Phase – 1.

I am grateful to the project coordinators, Prof. Charanraj B R, Prof. Sylvia Nancy and Prof. Sunitha R for organizing, managing, and helping with the entire process.

I take this opportunity to thank Dr. Shylaja S S, Chairperson, Department of Computer Science and Engineering, PES University, for all the knowledge and support I have received from the department. I would like to thank Dr. B.K. Keshavan, Dean of Faculty, PES University for his help.

I am deeply grateful to Dr. M. R. Doreswamy, Chancellor, PES University, Prof. Jawahar Doreswamy, Pro-Chancellor – PES University, Dr. Suryaprasad J, Vice-Chancellor, PES University for providing me various opportunities and enlightenment every step of the way. Finally, this project could not have been completed without the continual support and encouragement I have received from my family and friends.

ABSTRACT

Cloud storage is now in heavy demand as most internet users have resorted to the cloud to store a lot of their personal information as well as data. The cloud also serves as an efficient backup allowing users to collaborate, retrieve as well as upload files at any time or place. Most people have blind faith in their respective cloud service providers that they ignore the integrity of the files uploaded. The cloud service providers are silent on guaranteeing the integrity of user's files.

These files which are uploaded by the user can be downloaded at any place or time but have no provisions as such to check for the integrity of these files. Files uploaded can easily be, intentionally or unintentionally, tampered or corrupted by means of various vulnerabilities on the cloud provider's side and there is no assurance provided by the cloud provider that the file they download is the authentic file that they uploaded.

Our goal is to develop a tool that will allow users to verify the integrity of the files with support for portability and availability. The system we intend to design will be capable of uploading files to the cloud and verifying the files that are downloaded from the cloud for authenticity, while making the interface user friendly as well as command line interface for experienced developers.

TABLE OF CONTENTS

| Chapter No. | Title | Page No. |
|--------------------|--|-----------------|
| 1. | INTRODUCTION | 08 |
| 2. | PROBLEM DEFINITION | 10 |
| 3. | LITERATURE SURVEY | 11 |
| 4. | PROJECT REQUIREMENTS SPECIFICATION | 21 |
| 5. | SYSTEM REQUIREMENTS SPECIFICATION | 22 |
| 6. | SYSTEM DESIGN | 27 |
| 7. | CONCLUSION OF CAPSTONE PROJECT PHASE - 1 | 38 |
| 8. | PLAN OF WORK FOR CAPSTONE PROJECT PHASE - 2 | 39 |
| | REFERENCES/BIBLIOGRAPHY | 40 |
| | APPENDIX A DEFINITIONS, ACRONYMS AND ABBREVIATIONS | 41 |

LIST OF TABLES

| Table No. | Title | Page No. |
|------------------|------------------------------|-----------------|
| 1 | Software Requirements | 26 |
| 2 | ER Diagram | 34 |

LIST OF FIGURES

| Figure No. | Title | Page No. |
|-------------------|---|-----------------|
| 1 | Merkle Tree | 12 |
| 2 | System Overview | 14 |
| 3 | Integrity Verification Time vs File size | 15 |
| 4 | Integrity verifier Tool | 15 |
| 5 | Resigning Blocks Signed By Revoked User | 16 |
| 6 | System Model - I | 17 |
| 7 | System Model - II | 20 |
| 8 | Deployment Diagram | 25 |
| 9 | Component Diagram Overall | 29 |
| 10 | Sequence Diagram Upload | 30 |
| 11 | Sequence Diagram Verify | 31 |
| 12 | Sequence Diagram Download | 32 |
| 13 | Master Class Diagram | 33 |
| 14 | Packaging Diagram | 36 |
| 15 | Deployment Diagram | 37 |
| 16 | Gantt Chart | 39 |

CHAPTER 1

INTRODUCTION

Cloud storage is an in demand service offered by various providers and accessed by almost everyone who has access to an active internet connection. Various formats of data are uploaded every single second of the day from pictures, documents, files to videos and presentations. Users place their faith believing that the cloud is a secure way to store all data. Cloud storage offers the best deal in terms of storing files as it accounts for portability and availability and it is a proven fact as all our files are available for access all the time and can be accessed from anywhere but when it comes to confidentiality there is no concrete evidence stating that the files are tampered or not. Cloud providers do not provide complete transparency with respect to the confidentiality of the files.

In today's world with the advent of cyber crime, bullying and ransomware circulating across the internet there are no guarantees for files on the internet. There is a high possibility that the files we upload may not be the same files we download. The file may have been tampered by various means used by hackers with malicious intent. So it is important to make sure that the file we retrieve from the cloud is our own authentic file and has not been tampered to inject malwares into our system.

Features of Cloud that have made it so popular:

1. Major selling point of cloud storage is availability and portability. Files uploaded to the cloud are available at all times and are permanently stored unless there has been a server outage in which case the files will not be accessible for a very short period of time which is in itself a very rare occurrence.
2. With the growing popularity of cloud services, as many people have adopted the cloud service they have become cheap and affordable to the common people. Most cloud service platforms provide a certain amount of free storage for all its users thereby attracting a lot of users who are in need of infinite amounts of storage and portability.

Layer 7 Integrity Check For Cloud Service Providers

There exists a lot of cloud service providers in the markets starting from giant corporations such as google, amazon and microsoft to not very well known organizations like mediafire and various other providers.

There is a sense of doubt when it comes to the files we retrieve from the cloud as to whether they are authentic or have they been tampered with. To identify whether the file has been tampered or not is not a simple process and has an extra overhead which results in the flow of data from the cloud to be less smooth due to extensive wait times from the verification of files to be downloaded. This process of verification can be optimised by making use of merkle trees as well as hashing algorithms.

CHAPTER 2

PROBLEM DEFINITION

Cloud service providers are deemed secure by default. Most users are oblivious to any tampering of their files if it occurs. With such high progression in algorithms and data structures it is a minimal requirement for all software users to expect their applications to be extremely fast and smooth to handle. Therefore many cloud providers are silent on the integrity of the files uploaded. There is no means to check for the integrity of the files we download from the cloud.

The project aims to provide users with a means to verify the integrity of the files they download with a minimal overhead to the existing procedure coupled with an easy to use interactive interface.

Cloud providers in general are not free from vulnerabilities or attacks. They form a major target for hackers to exploit. Hence verification of integrity of files is of paramount importance. With the spread of malwares, worms and ransomwares increasing exponentially every year integrity of files cannot be taken lightly. More often than not the service providers are affected by ill educated users or employees who fall victim to phishing attacks. Therefore the cloud providers cannot claim complete security based on their existing api's. Therefore checking integrity of files downloaded from the cloud is a valuable addition to the existing software services provided by the cloud storage providers.

CHAPTER 3

LITERATURE SURVEY

In this chapter, we present the current implementations of the existing integrity checks in cloud.

3.1 Native Implementations

This section details papers which presented methods to perform integrity checks on the native cloud platform. These implementations are part of the cloud implementations and do not talk about providing integrity checks on top of existing cloud platforms.

3.1.1 HAIL: A High Availability and Integrity Check for Cloud Storage

HAIL achieves integrity check and retrievability of files through a distributed cryptographic solution for a set of servers. Goal of HAIL is to ensure resilience against mobile adversaries, which has the potential to corrupt all the servers during the system lifetime assuming not all servers are corrupted in a time step. If a server is found to be corrupted when a client performs integrity check, the file is reconstituted by the redundancy in other intact servers.

3.1.2 Iris: A Scalable Cloud File System with Efficient Integrity Checks

Iris is a practical, authenticated file system designed to support workloads from large enterprises storing data in the cloud and be resilient against potentially untrustworthy service providers. As a transparent layer enforcing strong integrity guarantees, Iris lets an enterprise tenant maintain a large file system in the cloud. In Iris, tenants obtain strong assurance not just on data integrity, but also on data freshness, as well as data retrievability in case of accidental or adversarial cloud failures

3.2 Application Layer Implementations

3.2.1 File System Support Solution

The main focus of this paper was to create a system that maintains the integrity of files hosted on the cloud by untrusted servers. They used a B+ tree which serves as a Merkle Tree to keep track of

files while maintaining a low tree depth. The verification of files uploaded happens by means of 32 byte hash. They also managed to integrate their implementation with dropbox.

Merkle Tree

Merkle tree is a tree in which:

- Leaf node is a hash of a data block.
- Non-leaf node is a hash of its children.

They are handy in distributed systems as they enable consistent data verification. They are efficient because they use hashes and size of hash is very small when compared to the file. They are also used in p2p networks like git, bitcoin, webTorrent etc.

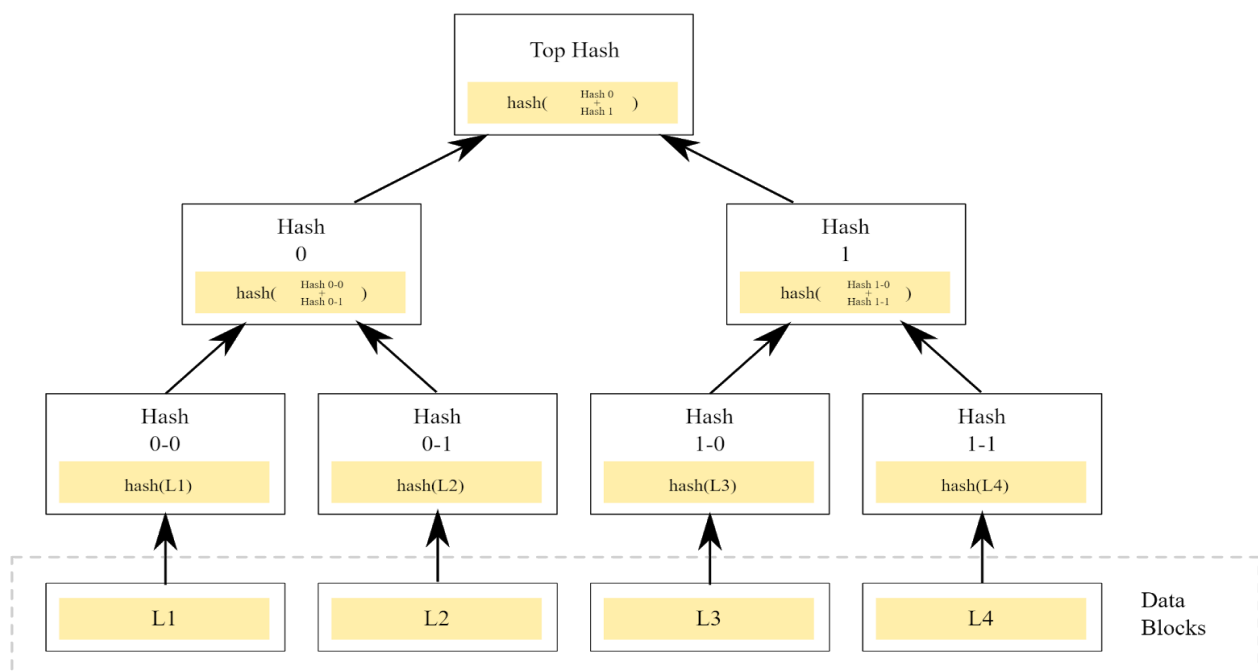


Figure 1 : Merkle tree

Analysis

1. The implementation described in this paper served as a file store on top of DropBox(Cloud Store Application) where dropbox does not provide complete transparency.
2. DropBox was augmented with a merkle tree to keep track of file and intermediate node hashes in the tree making it the only component that had to be kept persistent on the client's local file system.
3. This system will be able to detect file tampering but will not be able to reproduce the original files from an adversarial server.
4. The paper lays deeper emphasis to file integrity and does not account for confidentiality and availability components of security in cloud systems.
5. This paper focuses on building a single user Dropbox client which builds a Merkle tree as metadata
6. The client is able to verify that files downloaded are the original files that were uploaded to Dropbox.
7. Usage of Merkle tree,ensures that the client only keeps the root hash in persistent memory, everything else is stored in the cloud.
8. Operations supported:
 - a. Add/Upload
 - b. Delete
 - c. Download
 - d. Edit
 - e. Verify
 - f. List
9. All operations support $O(n \log n)$ time complexity.

3.2.2 Design and Implementation of an Efficient Tool to Verify Integrity of Files Uploaded to Cloud Storage

The main focus of this paper was to create a tool which is used to verify the integrity of files uploaded to file storage services. Digital signatures and Merkle Hash Tree (MHT) using SHA-256 and have been used for achieving the purpose.

They have integrated it with Dropbox. It has to be noted that they haven't supported dynamic update of data.

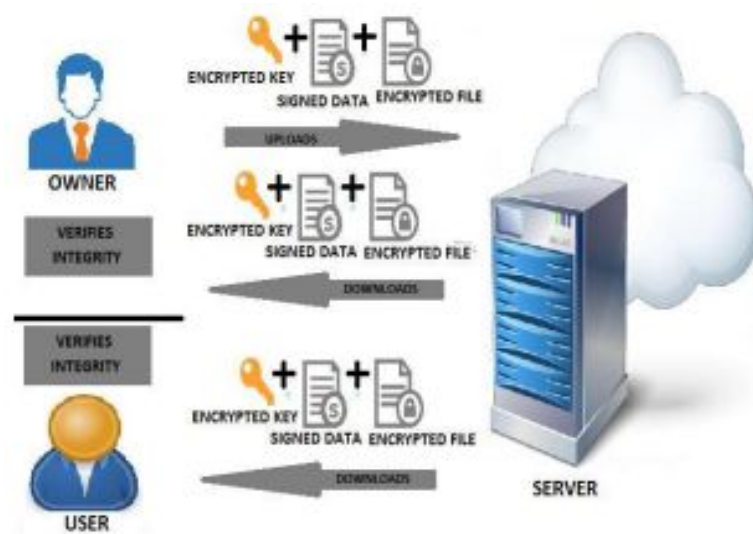


Figure 2 : System Overview

Analysis

1. Using the RSA algorithm, public and private keys are generated.
2. Using the AES-256 algorithm, the files are encrypted. Symmetric key is used for this.
3. Using the RLWE algorithm, key K is encrypted and saved in a file.
4. MHT is generated for the encrypted file and hashed nodes are converted to hex format and which is saved in a file. This file is then signed using a private key.
5. The owner uploads the following to the cloud :
 - a. Files (encrypted)

- b. Key K (encrypted)
 - c. Merkle Hash Tree file (signed)
6. Signature verification is done on an MHT file and if it succeeds, it indicates that the MHT file hasn't been tampered.
7. The user then compares the uploaded MHT file with the calculated MHT file. If they are the same, it indicates that the file has not been tampered with and proceeds for decryption.

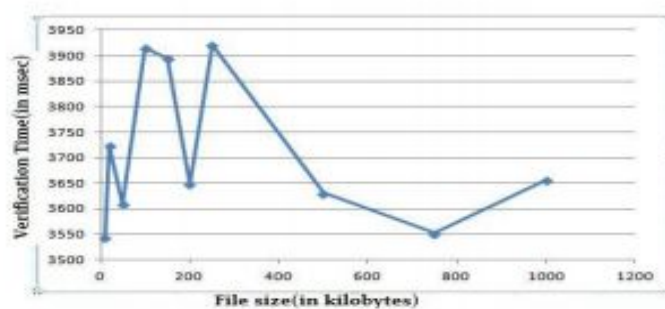


Figure 3 : Integrity verification time Vs file size



Figure 4 : Integrity verifier tool

3.2.3 PANDA: Public Auditing for Shared Data with Efficient User Revocation in the Cloud

1. The main focus of this paper:
 - a. to create an auditing mechanism for shared data.

- b. implement a very efficient user revocation mechanism.
2. Following concepts have been used:
 - a. Digital signature.
 - b. Proxy re-signature scheme.
3. Users in the group can download, modify and access shared data. The cloud resigns the data blocks that were signed by revoked users by utilizing the concept of proxy re-signature which thereby prevents serious amounts of computation during user revocation.

Analysis

1. One of the users in the group signs the shared data block and respective blocks are associated with their signature. Initially, the original user computes the signatures on shared data.

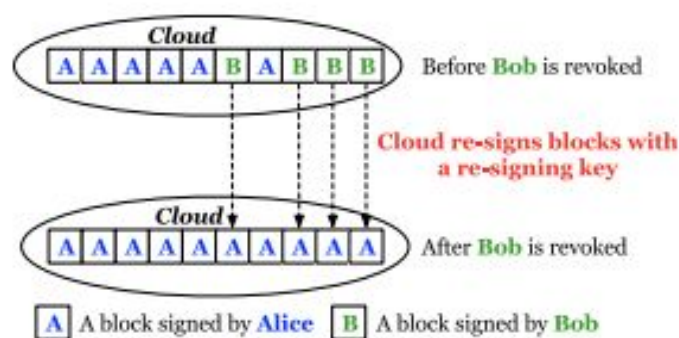


Figure 5 : Resigning blocks signed by revoked user.

2. The user signs the block using his private key whenever it is modified. Thus different blocks are signed by different users.

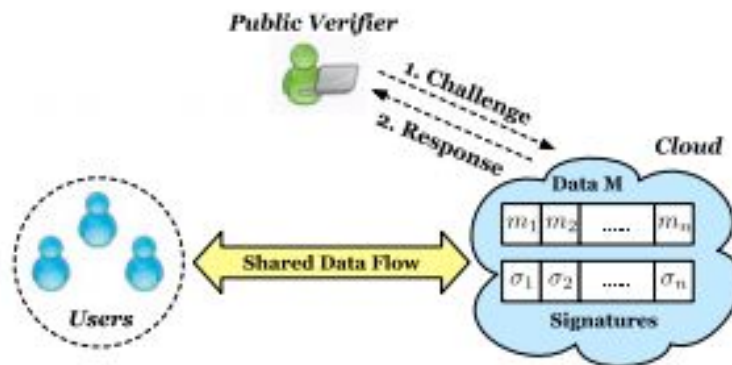


Figure 6 : The system model

3. Proxy Re-Signatures

- a. Signatures of two users are translated by the proxy. It basically converts a one user's signature into the signature of another user. It has to be noted that the proxy does not learn any signing key.

4. Efficient and Secure User Revocation

- a. The straightforward method is to allow an user to download the data and re-sign it. Since the data size can be high, it is highly inefficient. The blocks that were signed by the revoked user are signed using the resigning key. Because of this mechanism, a serious amount of computation is prevented.

5. Panda includes six algorithms as follows:

- a. KeyGen : Public key and private key are generated by the users in the group.
- b. ReKey : Re-signing keys are computed for each pair of users.
- c. Sign : Signature is computed when the user creates/modifies data blocks.
- d. ReSign : Cloud re-signs the data blocks which were signed by the revoked user.
- e. ProofGen : Proof for possession of data is generated.
- f. ProofVerify : Verification of data integrity is performed via verifying the proof responded by the cloud.

Alternate approaches in similar lines:

1. A private key can be shared among all users of the group which can be used to sign each data block. A new private key is securely distributed whenever the user is revoked and all the blocks have to be re-signed using this new private key. It decreases the efficiency of user revocation and also private key has to be securely stored.
2. Private keys of all the users could be stored on the cloud. This will allow re-signing the blocks without the need to download. But uploading the private keys on the cloud is a serious security issue and it has to be looked upon as we consider the cloud to be semi trusted.

3.2.4 Blockchain-Based Public Integrity Verification for Cloud Storage against Procrastinating Auditors

This paper proposes a certificateless public verification scheme against procrastinating auditors (CPVPA) by using blockchain technology.

The key idea is to require auditors to record each verification result into a blockchain as a transaction. Since transactions on the blockchain are time-sensitive, the verification can be time-stamped after the corresponding transaction is recorded into the blockchain, which enables users to check whether auditors perform the verifications at the prescribed time.

Moreover, CPVPA is built on certificateless cryptography, and is free from the certificate management problem.

Analysis

1. Public verification techniques enable users to outsource the data integrity verification to a dedicated third-party auditor. The auditor periodically checks the data integrity, and informs the users that the data may be corrupted once the checking fails.

2. The auditor is assumed to be honest. It is possible that the auditor generates a good integrity report to reduce verification cost or could collude with the cloud servers to maintain their reputation.
3. To ensure the security in the case that the auditor is compromised, the users are required to audit the auditor's behaviors.
4. Most public verification schemes are built on the public key infrastructure (PKI), where the auditor needs to manage the user's certificate to choose the correct public key for verification. Consequently, these schemes suffer from the certificate management problem including certificate revocation, storage, distribution, and verification, which is very costly and cumbersome in practice.

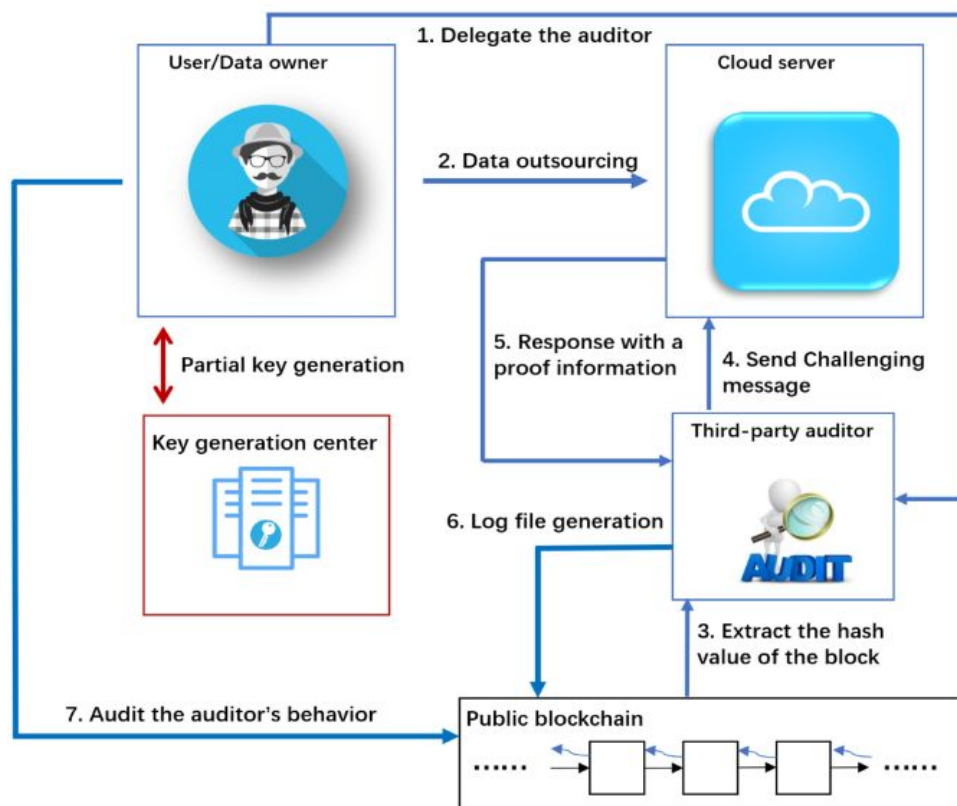


Figure 7 : System model

CPVPA, which resists malicious auditors and procrastinating ones without introducing any trusted entity, where each verification performed by the auditor is time-stamped by integrating it into a transaction of blockchain-based currencies, e.g., Ethereum. Such mechanism enables the user to check the time when the auditor performs the verification.

CHAPTER 4

PROJECT REQUIREMENTS SPECIFICATION

4.1 Data

Our product does not need any explicit data to analyse but only the cloud credentials of the user to login to their cloud account. The product deals with verifying the integrity of the data downloaded from the user's cloud. A User uploads a file to the cloud and a certain amount of metadata is generated for each file while uploading and stored on the user's cloud and the root hash value of the uploaded file is stored on the local file system. This data stored on the local file system is vital for the integrity check.

CHAPTER 5

SYSTEM REQUIREMENTS SPECIFICATION

5.1 Introduction

Cloud service providers are one of the main reasons why the majority of the internet users access the internet in the first place. Cloud providers are so popular primarily because of portability, availability and the assumption that the cloud is secure. Most cloud providers are silent on the integrity of the files that are available for download on their storage service. Our project aims at mitigating this sense of uncertainty in cloud services by allowing for fast and efficient integrity checks on cloud systems. The document establishes all the requirements of our project as well as the constraints that we have encountered.

5.2 Project Scope

This work aims to make it easier for layman users to verify the integrity of files they download from the cloud service providers. The project seeks to provide an interface for users to upload and download files and run integrity checks on the files they download from the cloud.

5.3 Features

5.3.1 Main Functions:

- The user is able to upload and download files through our application.
- The user is able to verify the integrity of the files downloaded.
- The user is able to check which file has been tampered with.
- The user is authenticated before giving access to download and upload files.
- The application generates hashes and creates node entries for all user uploads.

5.3.2 Additional Functions:

- Provide an easy-to-use interface for the user

5.3.3 User Classes and Characteristics

- End User: Upload/download files from cloud storage platform

5.3.4 Operating Environment

Hardware:

- Capable of running on a standard x86-64 machine

Software:

- Host OS: Independent
- Software Components:
 - Python

5.4 General Constraints, Assumptions and Dependencies

5.4.1 Constraints

1. Single user interface
2. Single cloud provider
3. Requires cloud credentials of the user
4. User initiates the integrity check

5.4.2 Assumption

1. Cloud Storage Platforms do not provide complete transparency.

5.4.3 Risks

1. Novelty of the problem chosen. No extensive work has been done on integrity verification till now.

5.5 Functional Requirements

1. The user is able to upload and download files through our application.
2. The user is able to verify the integrity of the files downloaded.
3. The user is able to check which file has been tampered with.
4. The user is authenticated before giving access to download and upload files.
5. The application generates hashes and creates node entries for all user uploads.

5.6 External Interface Requirements

5.6.1 User Interfaces

Login Portal

- A Login Portal used to authenticate a user.
- Authenticated users will be able to make use of the application.

File Upload

- The Upload portal will allow the user to select a file to upload.
- A Drag and drop option will be available as well.
- Cancellation of upload will be provided to the user by means of a button.

File Download

- Users will be able to select a file to download based on the click of a button.
- The user will be alerted if in case the file has been tampered with.
- A pop up message warning the user as to whether he wants to go through with the download.

5.6.2 Relative Timing of Inputs and Outputs

- The file uploaded is immediately reflected on the cloud service.
- File to be downloaded is verified and downloaded. Depends on the internet connectivity as well as size of file.

5.6.3 Error Messages

- Failed authentication
- Cloud Server connectivity error
- API key error
- Internet connectivity issues

5.6.4 API:

- Provides all software functionalities
- Can be accessed via endpoints provided through the application
- Can be accessed through a CLI.

5.7 Non Functional Requirements

Performance Requirements:

- Performance : time taken for verification, file upload and download should be optimum
- Reliability : detection of breach in integrity
- Interoperability : between cloud platforms and the software
- Portability : usability of the software on different OS

Safety Requirements:

- Access tokens of the user needs to be safely cached and needs to be removed after a certain duration to prevent non-users from accessing previous user's data.

Security Requirement:

- Confidentiality : only authorized users can access their respective files
- Availability : the user should be able to perform the integrity check at any given time
- Integrity: Files required by the application should be encrypted and stored securely and should not be easily accessible for common users.

5.8 Hardware Requirements

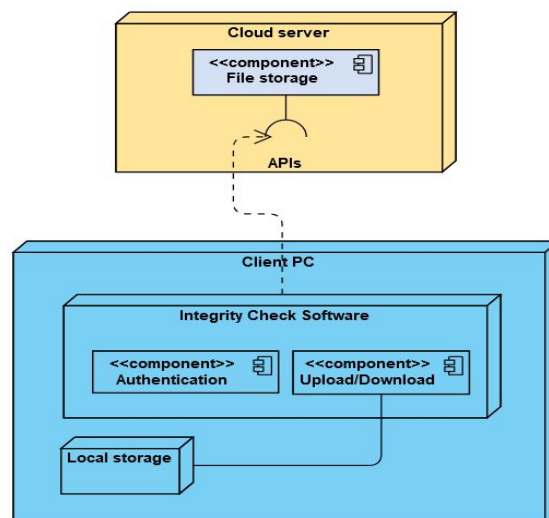


Figure 8 : Deployment Diagram

The solution we propose is a software that runs on the client system and stores the roothash value on the user's local system. The software performs integrity checks on files downloaded from a remote untrusted cloud server. The software can be built and run on any OS that supports Python 3.8 and above.

5.9 Software Requirements

| Name | Description | Version | Type |
|------------------|------------------------|------------|------------------------------|
| Cloud Services | Cloud storage Provider | NA | Cloud Storage Platform |
| Operating System | Platform Independent | NA | Operating System |
| Python | Programming Language | ≥ 3.6 | Programming Language Runtime |
| MongoDB | Database | ≥ 3.6 | Database |

Table 1 : Software Requirements

5.10 Communication Interfaces

All communications occur over HTTP/HTTPs requests

5.11 Safety Requirements

- Access tokens cached in the local file system must be kept secure, encrypted and should not be accessible to a common user. It can be a hidden file.

5.12 Security Requirements

- Users will log into his/her cloud service provider by providing his access keys to the application.

CHAPTER 6

SYSTEM DESIGN

6.1. Introduction

The title “Layer 7 integrity check for Cloud Service Providers” originated from the fact that our product operates in the application layer (7th layer) of the OSI model and also performs an integrity check on the files that are downloaded via the product. Although cloud providers promise a more secure and reliable environment to the users, the integrity of data in the cloud may still be compromised, due to the existence of hardware/software failures and human errors. We aim to Create a system/tool that maintains the integrity of files that are hosted on the cloud by untrusted servers. The user can verify that files he downloaded are the files that he uploaded and verify that they haven’t been tampered with.

6.2. Current System

Most cloud providers account for security in terms of availability and confidentiality while they provide substantial evidence for availability based on the fact that our files are accessible most of the time but do not provide substantial evidence for integrity. As all the encryption mechanisms happen in the background, the private keys are stored on the cloud itself and hence does not provide concrete proof for integrity.

Cloud service providers are generally silent on ensuring the integrity of the uploaded files on their cloud. No extensive work has been done on the same. Cloud service providers do not provide users means to check for integrity of the files uploaded.

6.3. Design Considerations

6.3.1. Design Goals

- The user is able to:
 - Upload and download files through our application.
 - Verify whether the integrity of the files downloaded is the same as that uploaded earlier.
 - Check whether files have been tampered.
- The application :

- o Authenticates the user before giving him access to upload/download files.
- o Generates merkle tree and creates node entries for all user uploads.
- o Fetches appropriate merkle tree nodes and generates the hash of the file(s) downloaded which will be further used to generate merkle tree.
- o Decides the integrity of files by comparing root nodes of merkle trees.

6.3.2. Architecture Choices: Client Server Architecture

A client server architecture enables us to run the product on the client's system and the cloud provider will function as the server. Client side of the application composed of a user interface and will facilitate users to upload and download files from their cloud storage.

Advantages of using this approach -:

- Assists Modular Development
- Simplifies the architecture

Disadvantages

- Performance is entirely dependent on client side system specification.

6.3.3. Constraints, Assumptions and Dependencies

Constraints:

- Single user interface
- Single cloud provider
- Requires cloud credentials of the user
- User initiates the integrity check
- Portability

Assumptions:

- Cloud Storage Platforms do not provide complete transparency.
- Users always make uploads only through their personal computer.

Dependencies:

- Open APIs of Cloud Storage Provider.
- User must provide his/her credentials.

6.4. High Level System Design

Component Diagram - Overall System

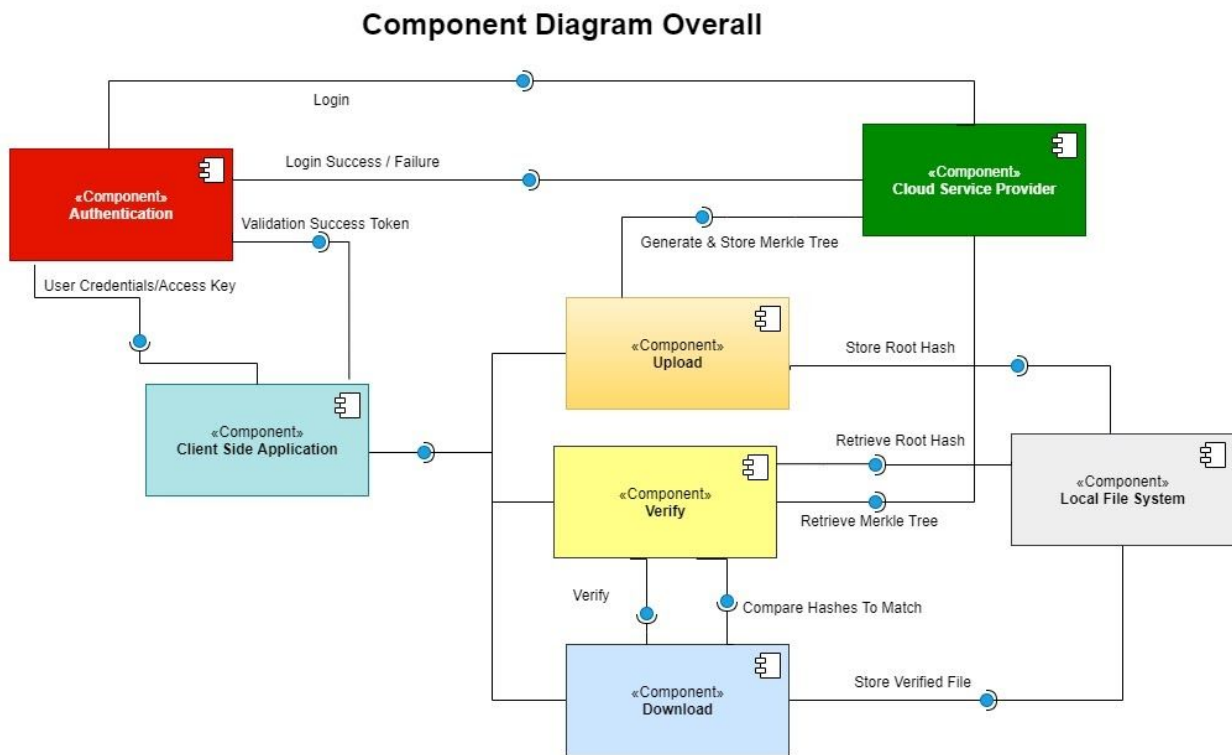


Figure 9 : Component Diagram Overall

- **Authentication** : The Authentication component ensures that the user is successfully authenticated and gains access to their respective cloud service.
- **Client Side Application** : The UI which facilitates the user to perform various operations which include uploading of file, downloading and verification for integrity.
- **Cloud Service Provider** : Used to maintain the merkle tree structure and retrieve as and when required by the Verify component.
- **Upload** : Used as a channel to upload files to the drive as well as to create all metadata required for processing later on.
- **Verify** : Used for processing the metadata to recompute merkle root nodes and compare with local copy of nodes.
- **Download** : Used to download the file and internally calls the verify api to make sure the file is not tampered.
- **Local File System** : Used to store the copy of the Merkle root node for comparison.

Sequence Diagram - Upload

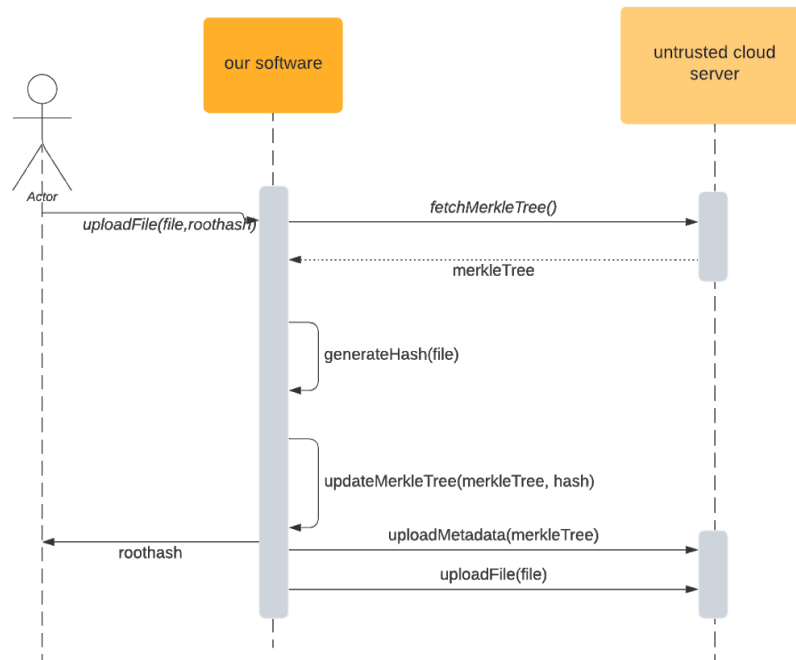


Figure 10 : Sequence Diagram Upload

- The user uploads the roothash and the file to the software.
- The software retrieves the merkle tree from the cloud storage.
- SHA-256 hash is generated for the new file to be uploaded.
- The merkle tree is updated with the new hash value.
- The new roothash value is returned to the user.
- The updated merkle tree and the file is uploaded to the cloud.

Sequence Diagram - Verify

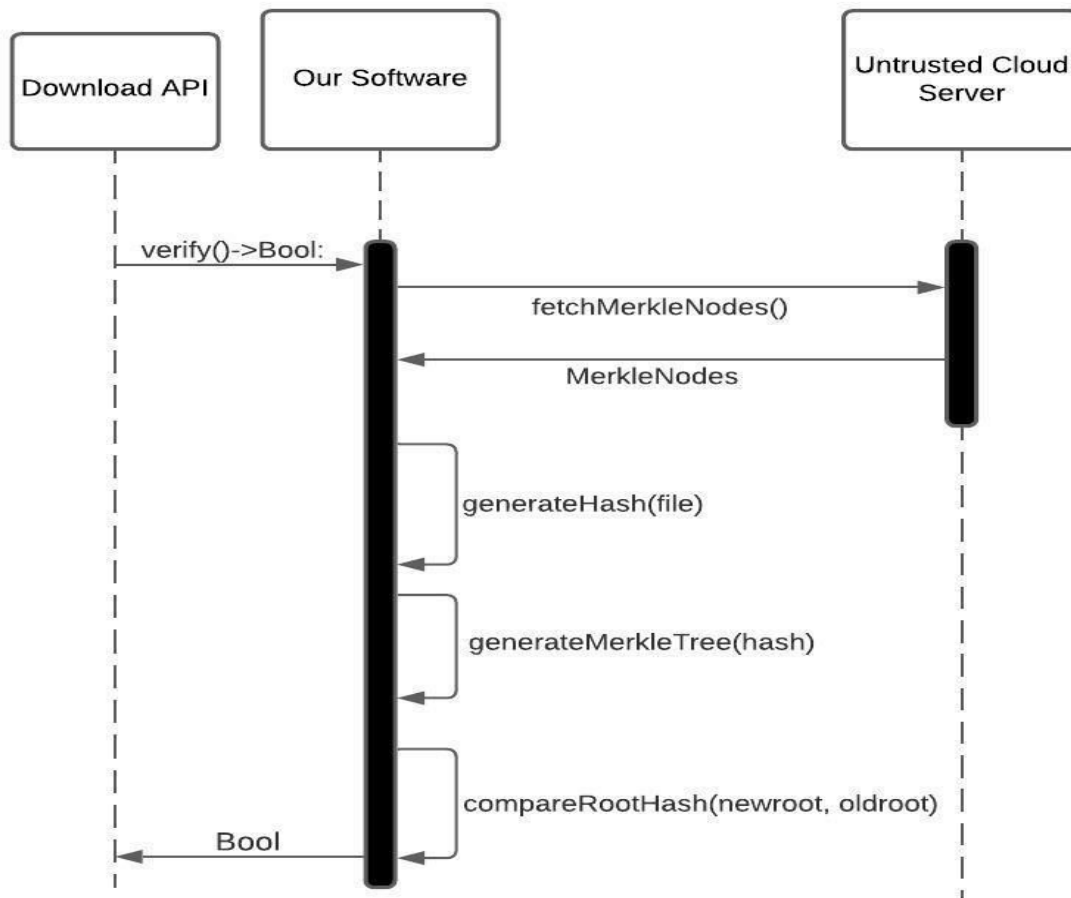


Figure 11 : Sequence Diagram Verify

- Verify API is hidden from the user and is made accessible only to download API.
- `Verify()` returns a boolean value which indicates whether the file has been tampered or not.
- `FetchMerkleNodes()` fetches the merkle tree nodes from the cloud to the local environment.
- `GenerateHash(file)` generates the hash of the file provided by the download API.
- `GenerateMerkleTree(hash)` generates the entire merkle tree using the hash of the file and appropriate merkle tree nodes.
- `CompareRootHash(newroot, oldroot)` compares for equality in merkle roots and returns a boolean accordingly.

Sequence Diagram - Download

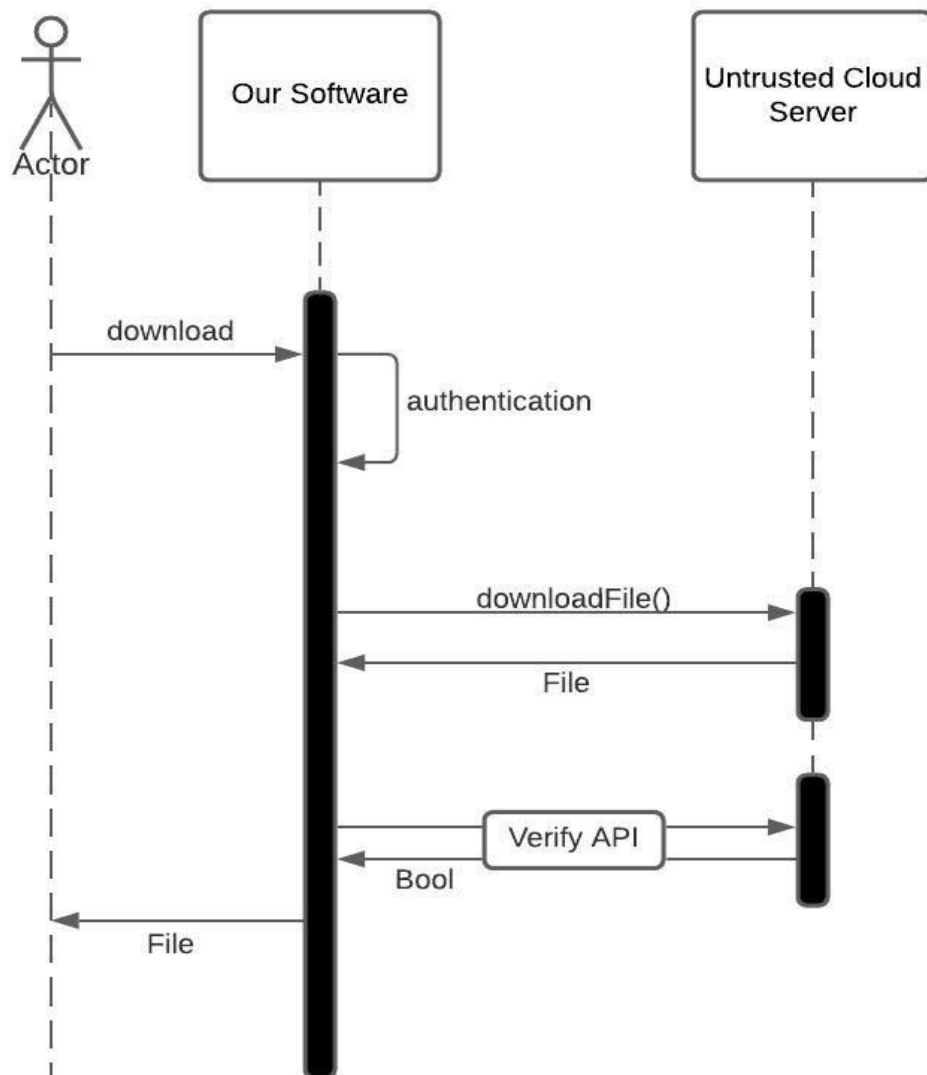


Figure 12 : Sequence Diagram Download

- Download API is exposed to the user and it calls verify API to verify the files downloaded for integrity.
- After authentication, DownloadFile() downloads the required file from the cloud.
- Calls the verify API to check for integrity. It returns a boolean indicating whether the file has been tampered or not.

6.5. Design Description

6.5.1. Master Class Diagram

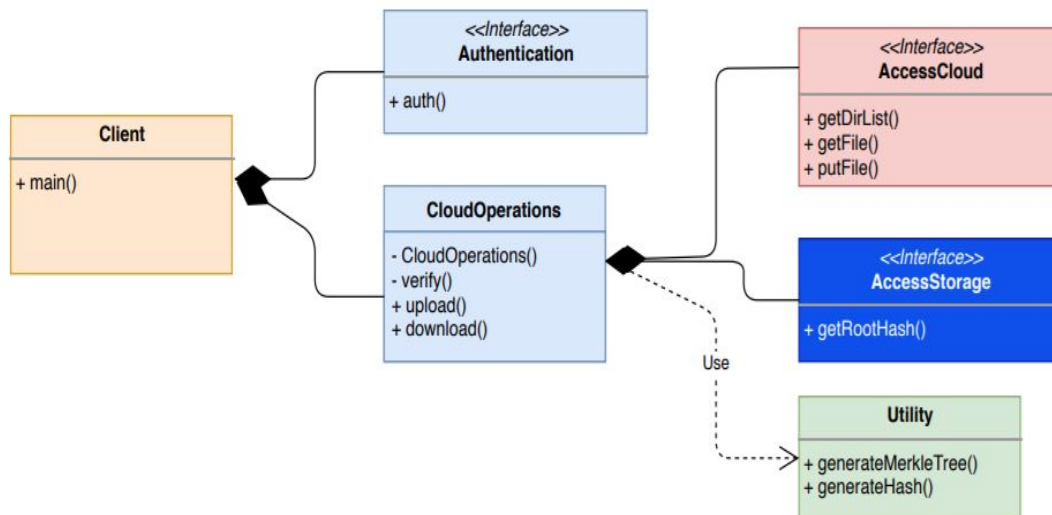


Figure 13 : Master Class Diagram

6.5.2. Reusability Considerations

Our application makes use of a few reusable components that include the following:-

- A cloud storage provider such as google and its pre existing api's.
- Sha 256 library of hash functions for hashing files.
- Merkle tree for storing the file system.
- Interfaces such as Authentication, AccessCloud and AccessStorage can be used to further implement various cloud storage providers.

6.6. Entity Relationship Diagram

| # | Entity | Name | Definition | Type |
|----------------------|-----------|---------------|--|--------------|
| ENTITIES | | | | |
| 1. | User | Client | Represents a client side user that interacts with the application interface. | User |
| # | Attribute | Name | Definition | Type (size) |
| DATA ELEMENTS | | | | |
| 1. | Name | Name | Identifier | String |
| 2. | Password | Auth Password | Authentication key specific to User. | Alphanumeric |

| # | Entity | Name | Definition | Type |
|----------------------|-------------|--------------|---|------------------|
| ENTITIES | | | | |
| 2. | File | File | File uploaded | User |
| # | Attribute | Name | Definition | Type (size) |
| DATA ELEMENTS | | | | |
| 1. | RootHash | RootNodeHash | Root Node of the merkle tree which is used for verification. | Sha 256 Hash |
| 2. | Merkle_Tree | Merkle_Tree | Merkle tree constructed to keep track of uploaded files and verify the files on download. | Merkle Tree Node |

Table 2 : ER Diagram

6.7. User Interface Diagrams

Login Portal

- A Login Portal used to authenticate a user.
- Authenticated users will be able to make use of the application.

File Upload

- The Upload portal will allow the user to select a file to upload.
- A Drag and drop option will be available as well.
- Cancellation of upload will be provided to the user by means of a button.

File Download

- Users will be able to select a file to download based on the click of a button.
- The user will be alerted if in case the file has been tampered with.
- A pop up message warning the user as to whether he wants to go through with the download.

Most of the above options will be a part of a command line interface followed by a GUI implementation if time permits.

6.8. Report Layouts

Our product does not generate any specific report but would indicate to the user that the file has been uploaded successfully or that the file to be downloaded has been tampered or not but does not account for retrieval of the tampered file if any. Our product does not say that integrity will be preserved but it assures the user that the file they receive is authentic but it does not add any extra features to ensure authenticity of files.

6.9. External Interfaces

The only external interface that the application will make use of is the Cloud Service Provider to store the uploaded files as well as its corresponding api's in order to handle the upload , download as well as verification of corresponding files.

6.10. Packaging and Deployment Diagram

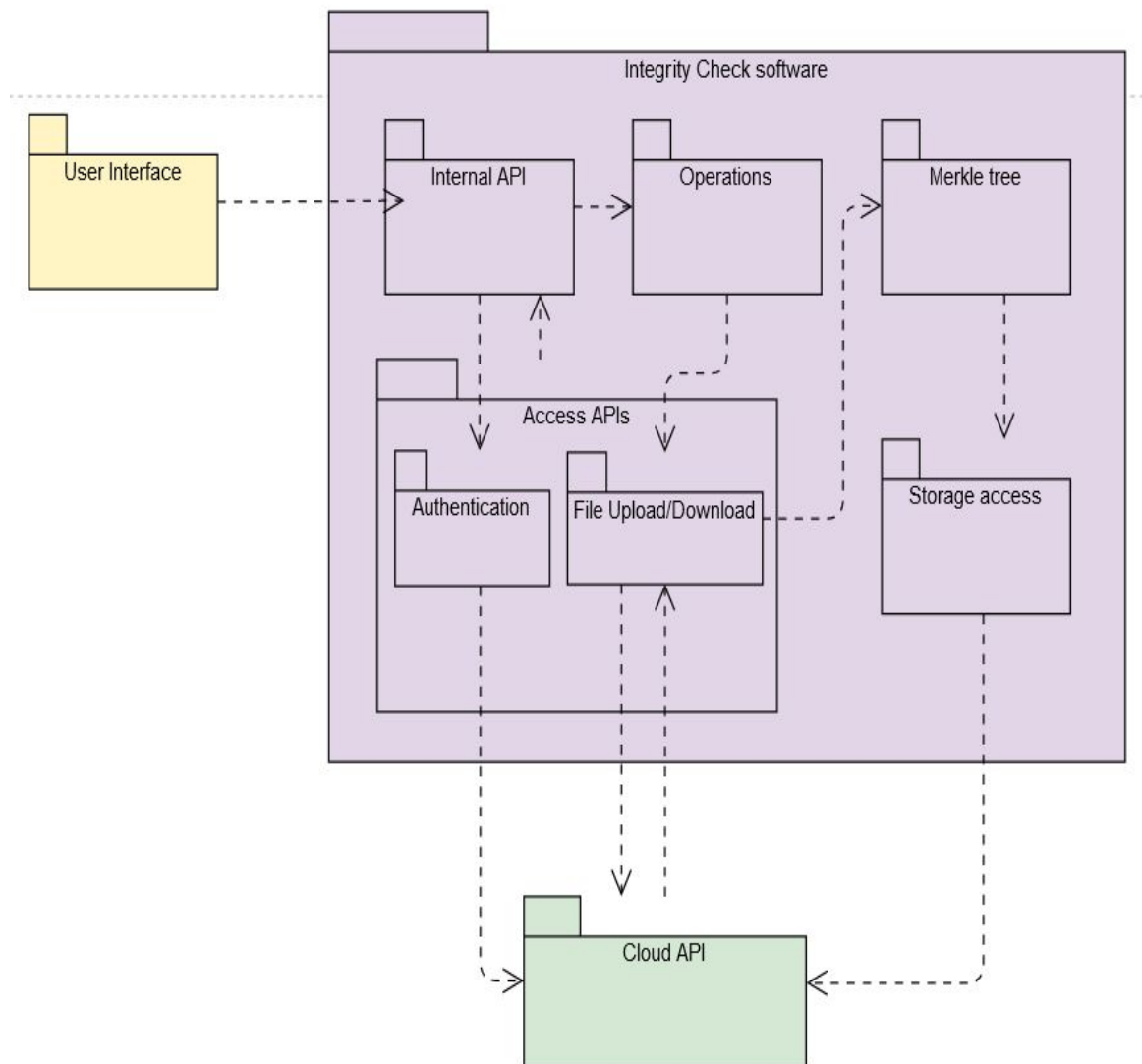


Figure 14 : Packaging Diagram

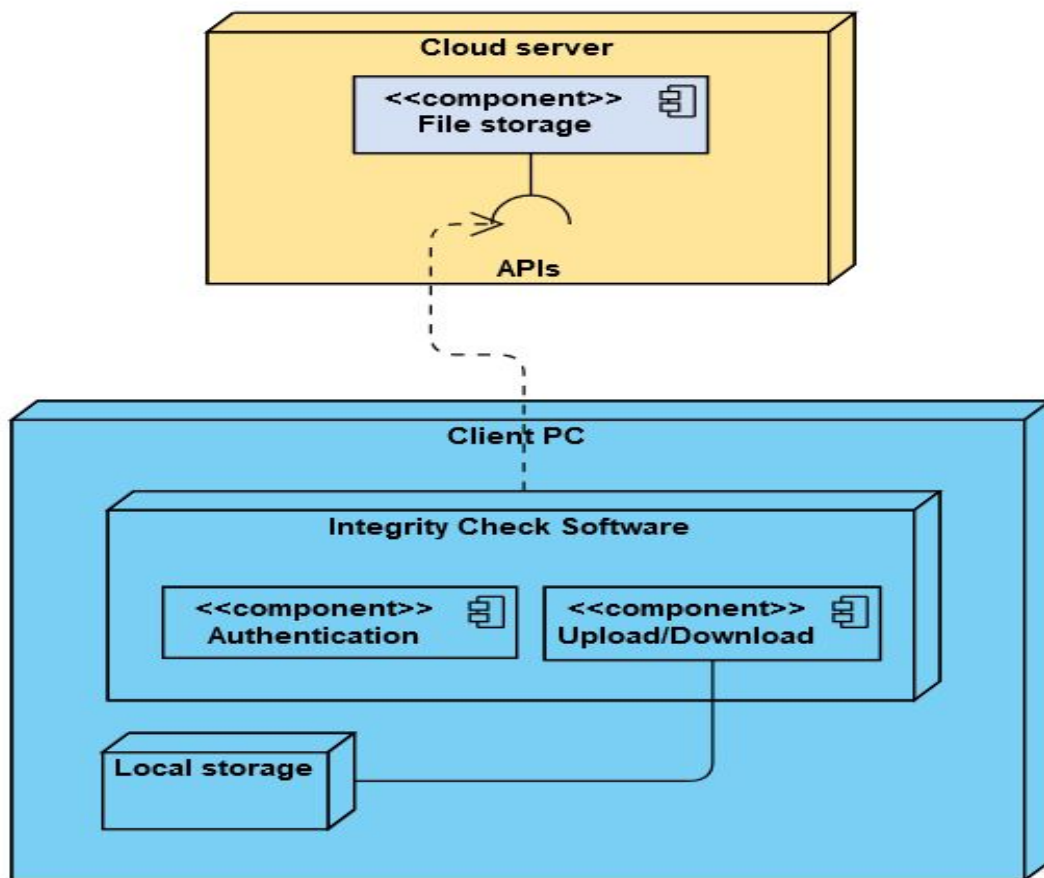


Figure 15 : Deployment diagram

6.11. Help

Documentation will be provided in the following formats:

- Existing online documentation for all the external API's
- READMEs for the structure as well as operation of all application endpoints
- In application suggestions as and when possible
- Community forum for addressing common issues

CHAPTER 7

CONCLUSION OF CAPSTONE PROJECT PHASE – 1

The literature survey and the high level design we managed to create during the course of this semester has yielded us with a lot of clarity regarding our software's end result, the look and feel of it as well as an implementation idea. We have laid a solid foundation for our project by means of the high level design and our literature survey has enabled us to establish a scope for our project by enlightening us with the various hurdles in different implementation procedures as well as what is possible and what is not.

We established that there are no existing cloud providers that provide users with api's to deal with integrity checks for the files they download from their own cloud service. Most cloud providers are silent on the integrity of the files uploaded / downloaded. We also were able to realise the need for an application that is able to add this functionality and we were able to realise its potential and impact on future cloud storage users.

We have decided upon most of our key implementation details for the project. We are eager to work on the Phase - 2 of the project and we are confident that we have the right tools, foundation and sufficient ground work in order to handle all the hurdles we face through the course of the next phase of the project.

CHAPTER 8

PLAN OF WORK FOR CAPSTONE PROJECT PHASE – 2

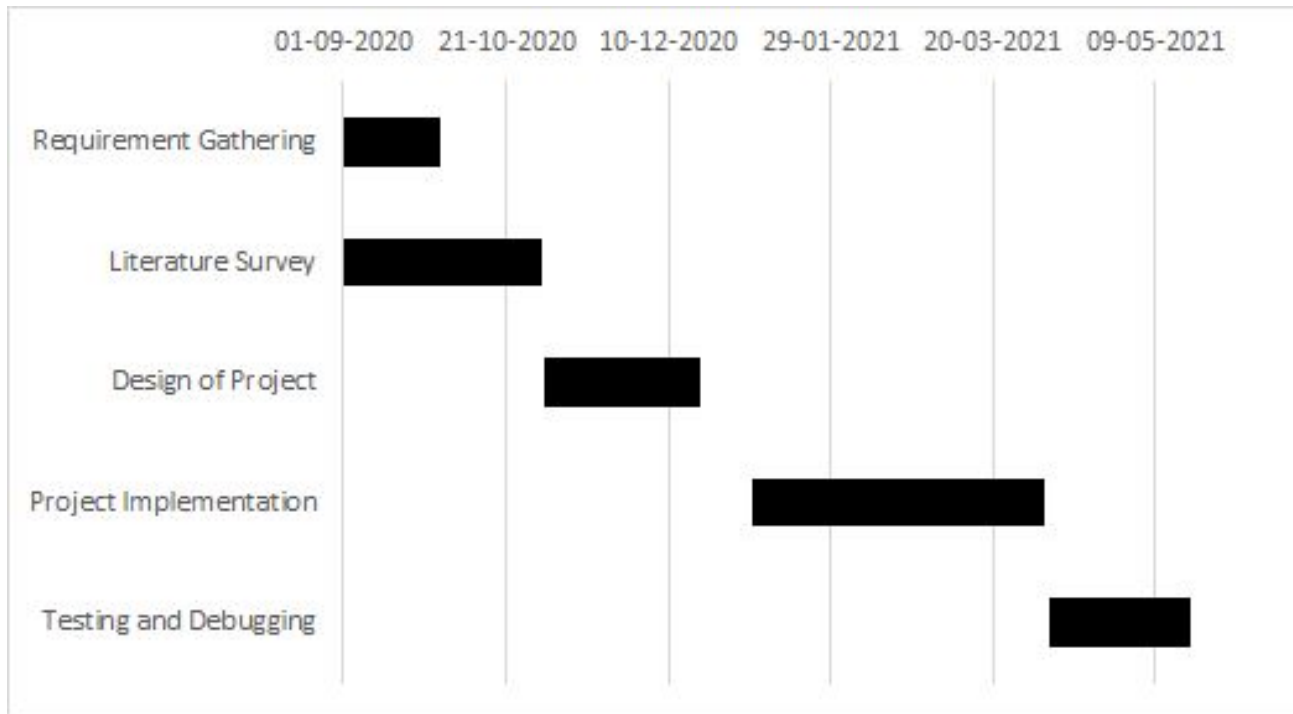


Figure 16 : Gantt Chart

Based on our extensive literature survey we have managed to identify various data structures and algorithms in order to design our application. We have also finalized on our high level design as well as a major amount of our key implementation details. Therefore we are sticking to the deadlines mentioned in the gantt chart above.

For the Phase - 2 of our project we intend to create a low level design and implement all the api's as mentioned in the high level design. We also intend to perform robust testing of all the endpoints before deployment as well as penetration test all the endpoints to make sure that our application has minimal bugs if not any.

REFERENCES/BIBLIOGRAPHY

1. Joannas, Msands, Brishima, "(File) System Support Solution".
2. Dalia Attas, Omar Batrafi, "Efficient integrity checking technique for securing client data in cloud computing".
3. Emil Stefanov, Marten Van Dijk, Alina, Ari Juels, "Iris: A Scalable Cloud File System with Efficient Integrity Checks"
4. FileZilla Pro : FileZilla Pro
5. Kevin D. Bowers, Ari Juels, Alina Oprea , "HAIL: A High-Availability and Integrity Layer for Cloud Storage"
6. Boyang Wang, Baochun Li, Hui Li, "Panda: Public Auditing for Shared Data with Efficient User Revocation in the Cloud".
7. [Packaging Diagram Tutorial](#)
8. [Deployment Diagram Tutorial](#)
9. [Sequence Diagram Tutorial](#)
10. [Cryptographic Hashing Functions -SHA-256](#)
11. [Cryptographic Hashing Functions - MD5](#)
12. [RSA \(cryptosystem\)](#)
13. [Symmetric key encryption](#)

APPENDIX A DEFINITIONS, ACRONYMS AND ABBREVIATIONS

Merkle Tree : Merkle tree is a tree in which every leaf node is labelled with the cryptographic hash of a data block, and every non-leaf node is labelled with the cryptographic hash of the labels of its child nodes.

Sha256 : Sha 256 is a cryptographic hash function whose size is exactly that of 32 bytes.