HIGH LEVEL DESIGN DOCUMENT

Layer 7 Integrity Check On Cloud Storage Platforms(L7ICCSP)

UE17CS490A – Capstone Project Phase – 1

Submitted by:

R Siva Girish	**PES1201700159**
Gaurav C.G	**PES1201700989**
Sangam Kedilaya	**PES1201701139**
Zenkar	**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)

PES University

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

TABLE OF CONTENTS

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.

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.

3. Design Considerations

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

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.

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:

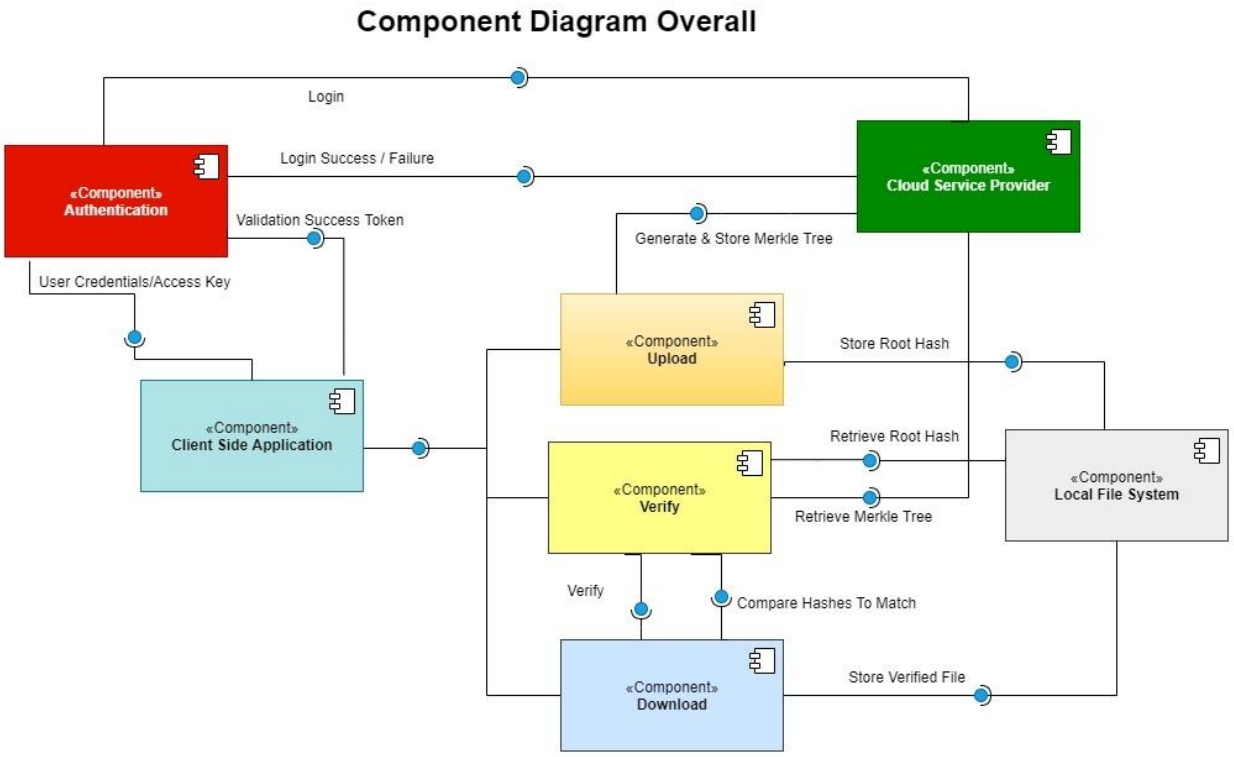
- Cloud Storage Platforms do not provide complete transparency.
- Users always make uploads only through their personal computer.

Dependencies:

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

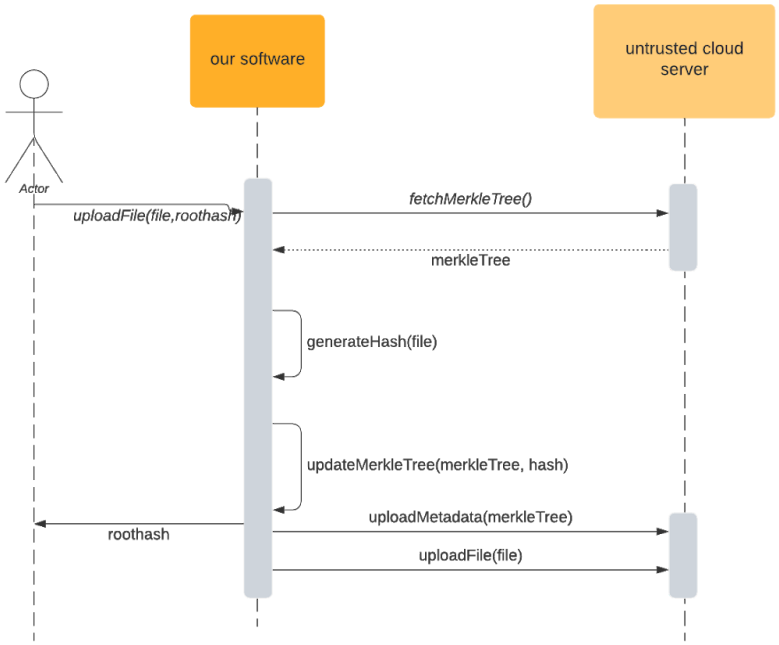
4. High Level System Design

Component Diagram - Overall System



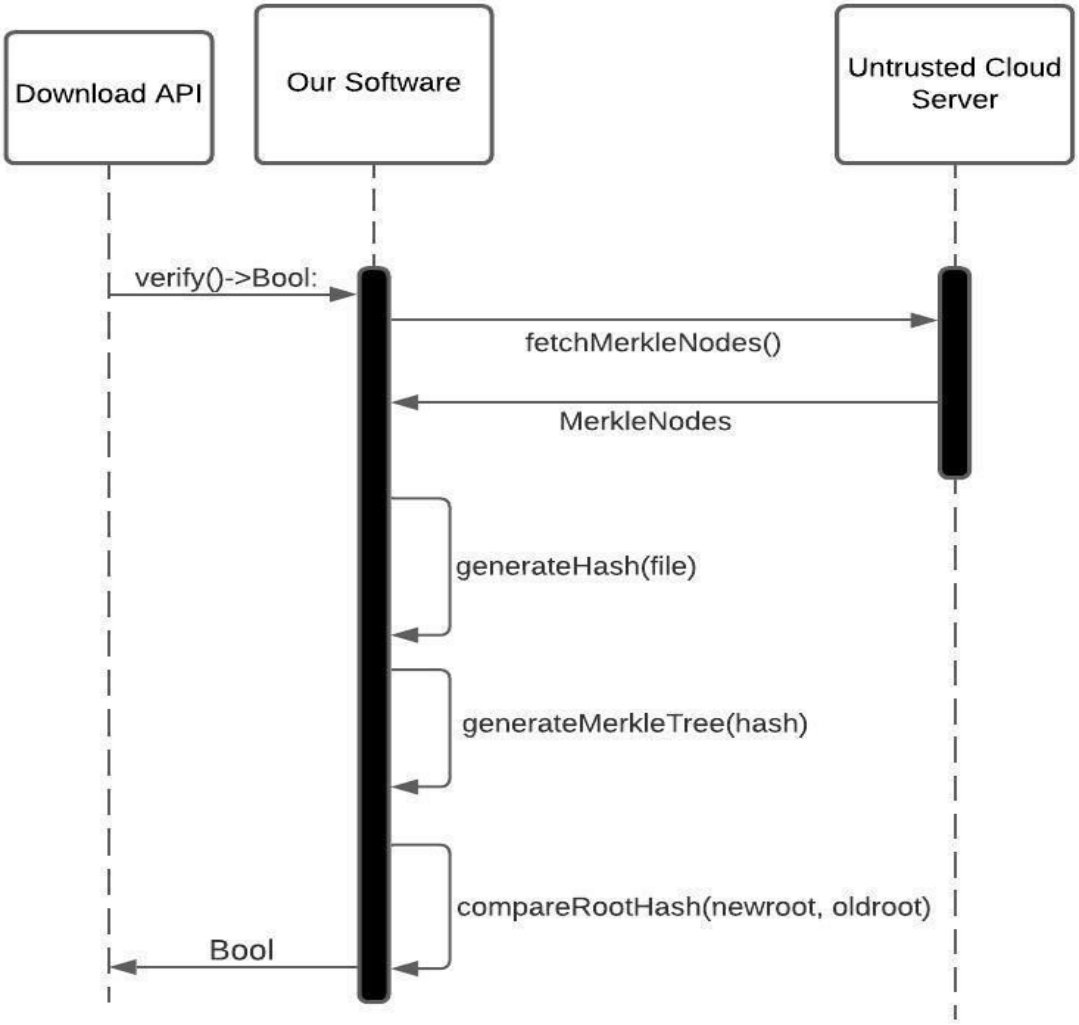
- **Authentication :** The Authentication component ensures that the user is successfully authenticated and gains access to their respective cloud service.
- **Client Side Application** : The UI which facilitates the user to perform various operations which include uploading of file,downloading and verification 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



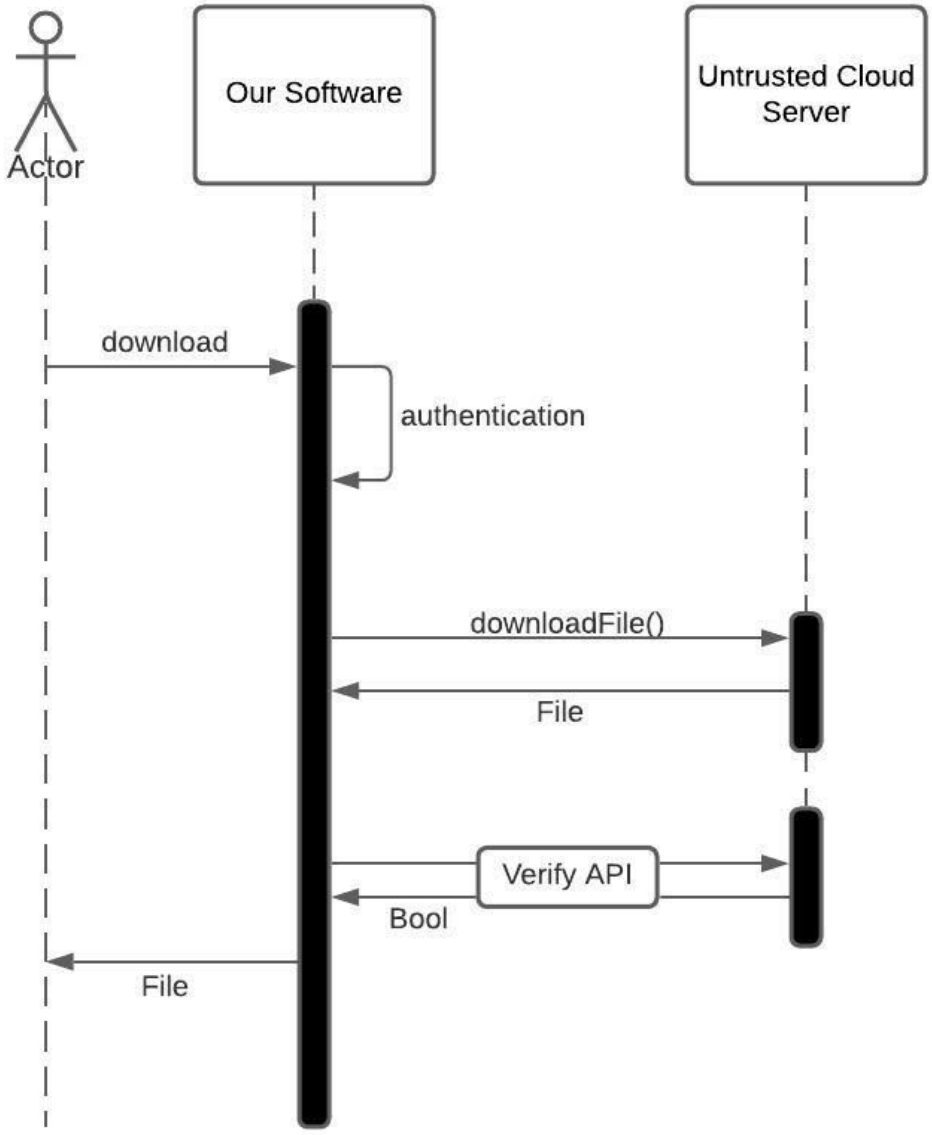
- The user uploads the roothash and the file to the software.
- The software retrieves the merkle tree from the cloud storage.
- SHA-256 hash is generated for the new file to be uploaded.
- The merkle tree is updated with the new hash value.
- The new roothash value is returned to the user.
- The updated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

Sequence Diagram - Verify



- 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.
- ComapareRootHash(newroot, oldroot) compares for equality in merkle roots and returns a boolean accordingly.

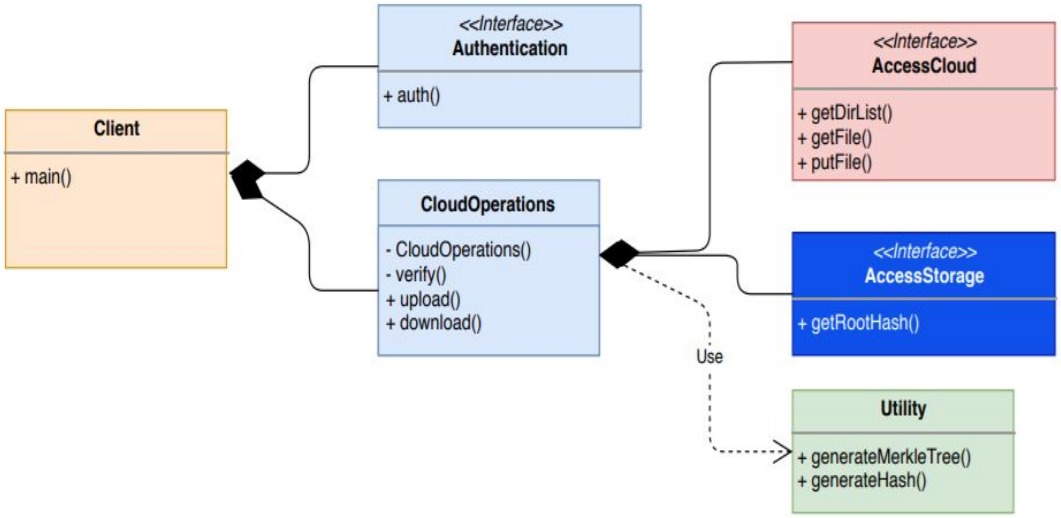
Sequence Diagram - Download



- 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.

5. Design Description

5.1. Master Class Diagram



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. 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

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.

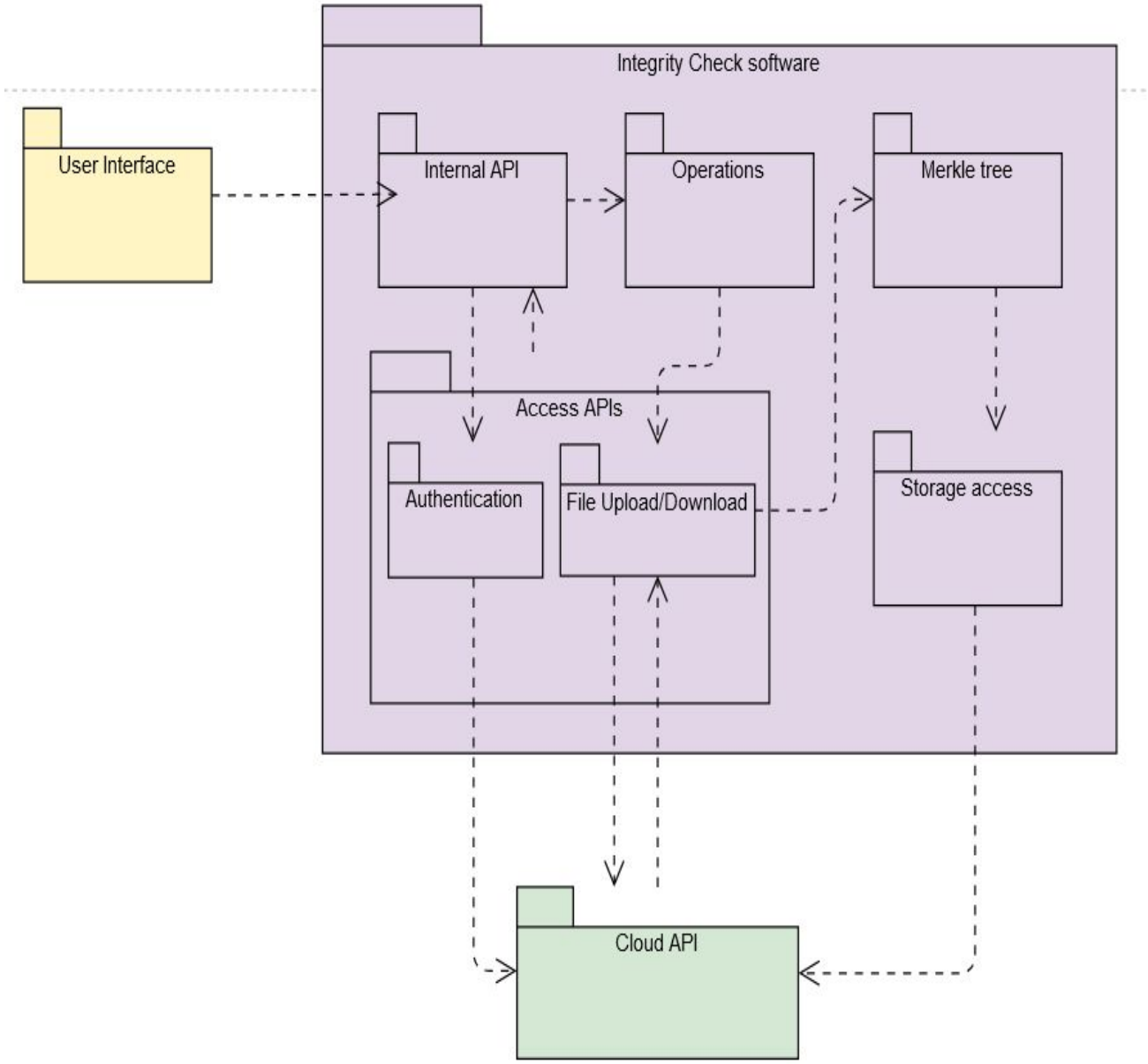
8. Report Layouts

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.

9. External Interfaces

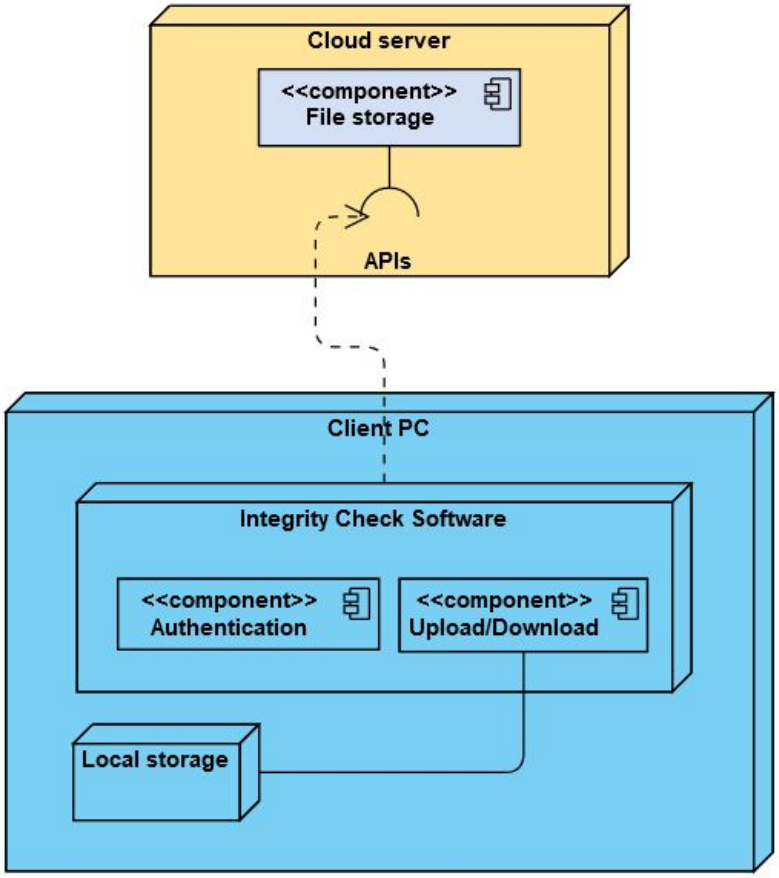
The only external interface that the application will make use of is the Cloud Service Provider to store the uploaded files as well as its corresponding api's in order to handle the upload , download as well as verification of corresponding files.

10. Packaging and Deployment Diagram



Packaging Diagram

Deployment diagram



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

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.

Appendix B: References

- Joannas, Msands, Brishima(File) System Support Solution.
- Dalia Attas, Omar Batrafi, "Efficient integrity checking technique for securing client data in cloud computing".
- Emil Stefanov, Marten Van Dijk, Alina, Ari Juels, "Iris: A Scalable Cloud File System with Efficient Integrity Checks"
- FileZilla Pro : FileZilla Pro
- Kevin D. Bowers, Ari Juels, Alina Oprea ,"HAIL: A High-Availability and Integrity Layer for Cloud Storage"
- Boyang Wang, Baochun Li, Hui Li,"Panda: Public Auditing for Shared Data with Efficient User Revocation in the Cloud".

Appendix C: Record of Change History

#	Date	Document Version No.	Change Description	Reason for Change
1.	**15 - 10 - 20**	**1**	Collection of Requirements	
2.	**13 - 11 - 20**	**2**	High Level Design	Portability Issues
3.	**18 - 11 - 20**	**2**	User Interface Diagrams	

Appendix D: Traceability Matrix

Project Requirement Specification Reference Section No. and Name.	DESIGN / HLD Reference Section No. and Name.
4. External Interfaces	9. External Interfaces
5. Non Functional Requirements	12. Design Details
3. Functional Requirements	4. High Level System Design